HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

GRADUATION THESIS

Hybrid Edge-Server Person Re-ID Scalable Microservices with Metadata-Enhanced Features

DUONG MINH QUAN
quandm.210710@sis.hust.edu.vn

Program: Data Science and Artificial Intelligence

Supervisor: Dr. Dang Tuan Linh _____

Signature

Department: Computer Engineering

School: School of Information and Communications Technology

HANOI, 06/2025

ACKNOWLEDGMENT

My four years at Hanoi University of Science and Technology are almost complete, a journey filled with both joy and a touch of sadness. More than just knowledge, what I truly treasure are the relationships I've built with friends, teachers, and fellow students.

To Mom and Dad, my deepest and most sincere thanks. You've always supported my choices unconditionally and given me everything to get to where I am today.

I also want to thank my supervisor, Dr. Dang Tuan Linh. Your careful guidance and clear direction were so important in helping me finish my papers, projects, and especially this graduation thesis.

To my friends who stuck by me through tough study sessions – especially my classmates, colleagues, and friends from the Academic Support Club – thank you for your support and companionship. A special shout-out to Huy Tuan, Le Huy, and Viet Anh for helping me with this thesis.

And finally, a heartfelt thank you to my girlfriend, Quynh Nga. Your constant support allowed me to focus and successfully complete many projects, including this thesis.

ABSTRACT

High costs and complex infrastructure requirements present significant barriers to adopting person Re-ID technology, particularly for organizations or small businesses with limited budgets or those lacking specialized AI computing hardware. This thesis addresses these limitations by proposing a novel hybrid edge-server architecture specifically designed for cost-effective person Re-ID deployment in such resource-constrained environments.

The proposed system distributes computational workload between lightweight, CPU-based edge devices and a centralized, GPU-enabled server, connected through an Apache Kafka message brokers cluster. Edge devices perform human detection using an optimized YOLOv11n model, achieving 39.5% mAP at 12 FPS on average on minimal hardware with 1 CPU core at 3.5 GHz and 512 MB RAM. The centralized server handles computationally intensive tasks including feature extraction, gender classification, and identity matching across multiple cameras.

A key innovation is the integration of metadata-enhanced retrieval optimization, where a custom-trained gender classification model with 95% accuracy reduces the search space during identity matching, improving retrieval speed by 20% compared to traditional approaches. The system is implemented using a containerized microservices architecture, enabling horizontal scaling and achieving up to 121 RPS for model inference utilizing Ray Serve with 99.5% system uptime.

This research contributes a practical, scalable solution that makes person Re-ID technology accessible to organizations with limited resources, enabling them to access surveillance applications in staffs management, security monitoring and operational efficiency without prohibitive infrastructure investments. The hybrid architecture proves that intelligent distribution of computational tasks can overcome the traditional cost-performance trade-offs in person Re-ID systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Definition
AI	Artificial Intelligence
API	Application Programming Interface
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
FnB	Food and Beverage
FPS	Frames Per Second
GPU	Graphics Processing Unit
HTTP	Hypertext Transfer Protocol
ID	Identification
IoU	Intersection over Union
mAP	mean Average Precision
MOT	Multi-Object Tracking
NMS	Non-Maximum Suppression
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
PoC	Proof of Concept
RAM	Random Access Memory
Re-ID	Re-identification
RPS	Requests Per Second
SOTA	State-of-the-Art
UAV	Unmanned Aerial Vehicle
YOLO	You Only Look Once

CHAPTER 1. INTRODUCTION

This chapter explores the significant financial constraints that prevent many organizations from implementing advanced solutions to enhance user experience, particularly those based on AI. This challenge forms the foundational motivation for this thesis: to develop a solution that enables organizations with limited resources to access and benefit from powerful technologies like person Re-ID.

1.1 Background and motivation

1.1.1 Background

In today's competitive market, many businesses face an extraordinary challenge in meeting evolving customer expectations, a hurdle felt most acutely by those operating on tight budgets. Customer expectations have fundamentally shifted from simple product transactions to demanding high-quality service provided by staff members. This change is particularly evident in sectors like Food and Beverage and retail, where 65% of customers report that staff responsiveness and personalized attention influence their purchasing decisions more than traditional advertising [1].

These financial constraints are a defining operational reality for many organizations, setting them apart from large corporations. In the FnB sector alone, 45% of businesses report that raw materials account for over 30% of their selling prices, leaving little room for major technology investments [2]. Furthermore, over 60% of FnB businesses have experienced revenue decreases while facing rising operational costs, including rent, labor, and materials [3].

This creates an "innovation deadlock" where these organizations:

- Recognize the critical need for better staff management and service optimization solutions.
- Understand that technology could provide competitive advantages in monitoring and improving staff performance, yet lack the capital to invest.

To help organizations with limited hardware resources still access Re-ID technology, they need solutions for surveillance problems like staff Re-ID to visualize activity maps and optimize service delivery. Modern AI technologies like Re-ID offer powerful solutions for understanding staff behavior patterns, monitoring service coverage across different areas, and identifying bottlenecks in staff allocation. Re-ID systems can seamlessly track staff movements across different sections of a store or restaurant, measure response times to customer needs, and identify areas where additional staff presence is required.

However, traditional Re-ID systems present significant economic barriers. Conventional implementations require expensive GPU-powered edge devices. When scaled across multiple cameras needed for comprehensive coverage, these costs become prohibitive. The high computational requirements of traditional Re-ID systems also demand powerful central servers for data processing and storage, further inflating the total cost of ownership. For organizations already struggling with thin profit margins, these substantial upfront investments often exceed their entire annual technology budgets.

1.1.2 Motivation

The challenges outlined above highlight a critical gap in the market: while Re-ID technology offers tremendous potential for staff monitoring and operational optimization, existing solutions remain financially inaccessible to organizations with limited resources. This creates an urgent need for affordable, scalable Re-ID solutions designed specifically for environments with limited hardware resources.

To address this need effectively, it becomes essential to examine existing architectural approaches and understand why they fall short for resource-constrained environments. The architectural design of a person Re-ID system is a critical consideration, with each approach presenting distinct trade-offs. The chosen architecture directly impacts the system's cost, scalability, and real-time performance, making it an especially important factor for organizations operating under budget constraints. The primary architectural models are **fully centralized**, **fully edge-based** and **hybrid**, yet none of these fully address the unique challenges faced by organizations.

To understand why current solutions fall short for organizations with limited resources, it is essential to examine each architectural approach and its specific limitations. The following subsections analyze these three primary models in detail:

a, Fully centralized architecture

The conventional approach involves a centralized architecture where standard cameras transmit their video feeds over a network to a single, powerful server. This central server is responsible for all computationally demanding tasks, such as person detection, feature extraction, and identity matching.

While this model simplifies on-site hardware, it introduces significant drawbacks that render it impractical for organizations with limited resources:

- **Network dependency and costs:** The continuous streaming of video from multiple cameras requires substantial network bandwidth, leading to high operational costs. The system's reliability is also contingent on network stability, as latency

or packet loss can result in incomplete data.

- **High server costs:** The computational load on the central server scales with the number of cameras, necessitating a significant upfront investment in high-performance server hardware. For many companies, this cost is prohibitive. This architecture also creates a single point of failure.
- **Management complexity:** A Re-ID pipeline consists of multiple processing stages. Managing this complex workflow for numerous concurrent video streams on a single machine presents a considerable technical challenge.

b, Fully edge-based architecture

To mitigate the network dependencies of the centralized model, an edge-based architecture places computational power on devices located near the cameras. These edge evices process video locally and transmit only lightweight metadata, such as feature vectors, to a central location.

This design reduces network bandwidth requirements and enhances resilience to network disruptions. However, it introduces its own distinct disadvantages:

- **High cumulative hardware cost:** The primary issue is the cost of the edge devices. While a single unit may be affordable, the expense escalates with each camera added, making large-scale deployments costly.
- **Distributed maintenance:** Managing a distributed fleet of edge devices is operationally more complex than maintaining a single server, increasing the burden of software updates and hardware troubleshooting.

c, Hybrid architecture

A hybrid architecture attempts to strike a balance by distributing the workload between edge devices and a central server. For instance, a low-cost edge device might handle initial person detection, while the more intensive matching tasks are offloaded to the server.

This approach aims to reduce hardware costs at the edge, but it presents its own set of complexities:

- **System integration challenges:** Dividing tasks creates a more intricate, multi-tiered system. Ensuring seamless communication and efficient integration between the edge and server components is a significant engineering task.
- **Potential for latency:** The handoff of data between the edge and the server can introduce processing delays. In applications requiring immediate responses, this latency can undermine the system's utility.

- **Workload balancing:** Achieving an optimal balance is difficult. If the edge device is underpowered, it can become a bottleneck. Conversely, if too much processing is offloaded to the server, the architecture reintroduces the bandwidth and cost issues of the centralized model.

In summary, the analysis of existing Re-ID architectures reveals a significant opportunity: while current solutions fail to meet the needs of resource-constrained organizations due to prohibitive costs, network dependencies, and operational complexity, the underlying technology holds immense potential for transforming staff monitoring and operational efficiency. This gap between technological capability and practical accessibility represents not just a challenge, but a compelling opportunity to democratize advanced surveillance technology.

The persistent "innovation deadlock" that prevents smaller organizations from leveraging Re-ID technology can be broken through thoughtful engineering and architectural innovation. By addressing the fundamental cost and complexity barriers, we can unlock the transformative potential of person Re-ID for organizations that need it most-those operating with limited resources but requiring maximum operational efficiency.

This thesis presents a feasible solution: a novel hybrid architecture specifically engineered to make Re-ID technology affordable, scalable, and manageable for surveillance systems in resource-constrained environments. Our approach promises to bridge the gap between cutting-edge AI capabilities and practical deployment constraints, enabling organizations of all sizes to harness the power of intelligent staff monitoring and operational optimization.

1.2 Objectives and scope of thesis

Section 1.1 presents various challenges that small organizations are facing due to their limited financial ability to implement solutions for internal surveillance. The first objective of this thesis is to design and implement a hybrid edge-server person Re-ID pipeline that can be deployed on resource-constrained devices, especially on CPU-based devices for edge deployment. This proves that GPU-based devices are not requisite for person Re-ID. This pipeline can detect, extract semantic vision features, and track person movements to report the results to the central server. The output of the pipeline can be used for various applications such as staff monitoring, service coverage analysis, and operational optimization.

Furthermore, to enhance the speed and accuracy of identity retrieval, a specialized gender classification model is proposed to be deployed on the edge device as a metadata enhancement component. This lightweight model is trained on a carefully

prepared dataset and achieves high classification accuracy while maintaining computational efficiency suitable. By incorporating gender information as metadata, the system can intelligently partition the search space during identity matching operations, significantly reducing computational overhead and improving retrieval performance. This metadata-driven approach not only accelerates the matching process but also enhances the overall system's scalability by reducing the workload on the central server during peak operations.

The scope of this thesis is limited to implementing a complete pipeline using Docker containerization, with resource constraints of 1 CPU core and 512MB RAM per edge device simulation. This configuration enables the creation of multiple virtual edge devices on a single physical machine for comprehensive testing and validation. Data collected from these simulated edge devices is transmitted to the centralized server through a Kafka message broker cluster, ensuring reliable and scalable communication. Both the Kafka cluster and centralized server are implemented as Docker containers, providing a deployment-ready solution that can be easily adapted for production environments with minimal network configuration adjustments.

1.3 Tentative solution

Based on the presented objectives in Section 1.2, the work composes of two main parts.

In the first part, based on researched technologies, the pipeline of the system will be proposed. The system will consist of three main components: edge devices with CPU-only computing capabilities, Kafka message broker cluster, and a centralized server. The edge devices will be responsible for human detection and basic preprocessing before sending data to Kafka message brokers. These lightweight edge devices eliminate the need for expensive GPU hardware while maintaining real-time processing capabilities. The Kafka message broker cluster serves as the communication backbone, ensuring reliable and scalable message delivery between edge devices and the central server. The third component is the centralized server, which combines multiple services following a microservices architecture including Model service, ID verifier service, vector database, and multiple consumers that perform feature extraction and identity matching using these services. The human detection model will be carefully selected among state-of-the-art models based on the criteria of accuracy and computational efficiency so that it can run effectively on CPU-only edge devices. This distributed architecture ensures optimal resource utilization while maintaining system scalability and cost-effectiveness.

The second part includes building a tailored, lightweight gender classification

model specifically for limited computing resources and integrating it into the Re-ID pipeline for metadata enhancement. A custom gender classification model will be developed using EfficientNet-B0 architecture and trained on the P-DESTRE dataset to achieve high accuracy while maintaining computational efficiency. This gender classification capability will be integrated into the system to enhance the retrieval process by reducing the search space during identity matching operations. A prototype will be deployed in practice for comprehensive testing purposes, validating the system's performance in real-world scenarios and demonstrating its viability for deploying on resource-constrained devices.

1.4 Contributions

1. An application is deployed on hybrid edge-server devices and uses a microservices architecture, allowing for easy system scaling (increasing the number of cameras). It includes:
   - An optimized Docker image for running the edge device processing flow with custom CPU and RAM configurations, tailored for deployment on CPU-only devices.
   - Optimized Docker images for deploying multiple model inference services through Ray Serve, including feature extraction and gender classification models, enabling high throughput and low latency for model inference via RESTful APIs.
   - A custom-trained, lightweight gender classification model with 95% accuracy for metadata enhancement.
   - A vector database optimization algorithm for efficient identity retrieval. This uses a person's metadata (gender) to reduce the search space, improving retrieval speed and accuracy.
2. This thesis also provides an interactive web application. It lets users monitor the system, view live camera streams.

1.5 Organization of thesis

The remaining of this thesis will be organized as follows:

Chapter 2 presents a comprehensive literature review and foundation theory essential for understanding the proposed system. The chapter is divided into two main sections: related works covering recent advances in person Re-ID, edge computing in AI, and microservices architectures; and foundation theory providing detailed technical background on object detection (YOLOv11), object tracking (ByteTrack), feature extraction (OSNet and LightMBN), image classification (EfficientNet-B0),

message queuing (Apache Kafka), model serving frameworks (Ray Serve), containerization (Docker), and vector databases (Redis). These investigations form the theoretical foundation upon which the proposed hybrid edge-server system is built.

Chapter 3 introduces the comprehensive methodology of the proposed hybrid edge-server person Re-ID system. This chapter presents the system overview and detailed implementation of three main components: the Kafka message broker cluster for reliable asynchronous communication, edge devices optimized for CPU-only human detection and preprocessing, and the centralized server managing microservices including model inference, vector database operations, and identity matching. The chapter also covers the deployment strategies, hardware implementations, and the integration of lightweight models specifically designed for resource-constrained environments.

Chapter 4 presents the experimental results and evaluation of the deployed system. This chapter covers the environmental setup including hardware specifications for both edge devices and the centralized server, dataset preparation and evaluation metrics, performance analysis of edge device processing capabilities, and evaluation of the custom gender classification model. The experimental results demonstrate the system's effectiveness in real-world scenarios and validate the viability of the proposed approach for deployment on resource-constrained devices.

Chapter 5, the final chapter, provides a comprehensive summary of the achieved results, discusses the contributions and limitations of the proposed system, and presents suggestions for future improvements and research directions.

CHAPTER 2. LITERATURE REVIEW

Chapter 1 established the foundational context by identifying the innovation deadlock faced by small organizations or businesses, defining research aims for affordable person Re-ID systems, and outlining the thesis contributions. This chapter presents related works in Section 2.1 and foundation theory in Section 2.2, where models, frameworks, and algorithms for the end-to-end pipeline are detailed. To achieve the thesis objectives, four key technical components are examined: lightweight object detection using YOLOv11 for CPU-based edge deployment (Section 2.2.1), efficient object tracking through ByteTrack algorithms (Section 2.2.2), optimized feature extraction and lightweight image classification for human metadata in limited hardware resource (Sections 2.2.3 and 2.2.4), and distributed system infrastructure including message queuing,
containerization, and vector database optimization (Sections 2.2.5, 2.2.7, and 2.2.8).

2.1 Related works

2.1.1 Person re-identification

Recent advances in person Re-ID have significantly enhanced performance across various scenario, primarily relying on powerful computational resources. In the context of visible–infrared ReID, Guo et al. (2025) introduced the Region-based Augmentation and Cross Modality Attention (RACA) model [4], which leverages region-level augmentation (PedMix) and a modality feature transfer (MFT) module with cross-attention to reduce interference between modalities, yielding notable improvements on SYSU-MM01 and RegDB benchmarks.

Addressing unsupervised learning, Qin et al. (2025) proposed Attention-based Hybrid Contrastive Learning (AHCL) [5]. Their framework integrates spatial and channel attention with a hybrid contrastive loss, combining cluster-level and instance-level representations to bolster ReID accuracy without labels.

Multimodal ReID has been further advanced by Yan et al. (2025) through FusionSegReID [6], which fuses image features, textual descriptions, and segmentation masks to enhance robustness-especially in occluded or low-quality scenarios. Interactive, language-driven retrieval was pushed forward by Niu et al. (2025) in ChatReID [7]. This framework uses a Vision–Language Model (LVLM) with Hierarchical Progressive Tuning, enabling interactive, VQA-style queries to improve identity-level matching performance.

Despite the impressive progress of person ReID, most state-of-the-art models

demand substantial hardware and computational resources, limiting their practicality on lightweight or embedded devices. To address this, **OSNet** (Omni-Scale Network) was proposed by Zhou et al. [8]. OSNet is a compact yet powerful architecture, specifically designed for efficient deployment. It employs *omni-scale feature learning*, utilizing multiple convolutional streams with varying receptive field sizes within each residual block to capture both fine-grained details and global features. Additionally, OSNet integrates a *Unified Aggregation Gate (UAG)* that adaptively fuses multi-scale features via channel-wise weighting. By leveraging depthwise-separable convolutions, OSNet significantly reduces computational cost and model size, achieving state-of-the-art performance despite being substantially lighter than standard models like ResNet-50.

Building upon OSNet's efficiency, LightMBN (Lightweight Multi-Branch Network) introduced by Herzog et al. [9] further optimizes this backbone architecture. LightMBN expands OSNet by adding specialized global, part-based, and channel-wise branches, thereby enriching feature representations without significant complexity increases. It also incorporates enhanced training techniques, including label smoothing, random erasing, and cosine learning rate schedules, leading to improved generalization performance. Consequently, LightMBN achieves impressive accuracies on widely-used benchmarks like Market-1501 and CUHK03, outperforming many heavier architectures while remaining lightweight and suitable for resource-constrained deployments.

Given their complementary strengths in efficiency and performance, OSNet and LightMBN form a robust backbone choice for deployment in lightweight Re-ID pipelines.

2.1.2 Edge computing in AI

Edge computing has emerged as a transformative approach in artificial intelligence, bringing computational resources closer to where data is generated and directly to end users. The global edge AI market size was valued at approximately USD 20.78 billion in 2024 and is expected to grow significantly, at a rate of 21.7% annually, from 2025 to 2030 [10]. This growth indicates a strong demand for real-time processing, lower latency, and improved privacy in various AI applications.

Within retail and customer experience contexts, edge computing provides substantial benefits. It enables retail IT teams to manage cloud expenses effectively by strategically selecting which data to send to the cloud, processing only critical information rather than all raw data [11]. This selective processing is particularly beneficial for person Re-ID systems, where enormous video data streams can be filtered and

analyzed locally, and only essential features are transmitted to centralized servers.

Recent research has also concentrated on adapting AI models specifically for edge environments. Novel person Re-ID methods incorporate pedestrian edge features directly into their representations and leverage these edge characteristics to enhance global context feature extraction [12]. Such methods highlight the practical feasibility of deploying sophisticated Re-ID algorithms on devices with limited computational capabilities.

Edge AI is becoming more widely available across many applications. Edge Intelligence, or Edge AI, means moving AI processing from cloud systems directly to edge devices where data is created. This change is important for making AI more available and affordable, especially for organizations that may find cloud-based AI solutions too expensive [13].

Therefore, this thesis leverages edge computing paradigms to distribute computational workloads across edge devices, enabling cost-effective deployment of person Re-ID systems while maintaining real-time performance requirements.

2.1.3 Microservices and distributed systems

Microservices have become increasingly popular for building scalable AI systems. In general, microservices focus on modularity, meaning each service handles a specific function independently. These services are loosely connected, easy to deploy individually, and can scale separately [14].

In person Re-ID systems, microservices offer clear advantages, especially when deployed across distributed environments. For example, using microservices in combination with AI-powered edge computing gateways helps handle privacy concerns while efficiently identifying the same person from multiple camera angles and locations [15].

However, using distributed setups in Re-ID introduces new challenges. Traditional Re-ID algorithms typically prioritize accuracy but aren't designed with distributed deployment in mind. Distributed environments demand algorithms that are lightweight and computationally efficient [16]. Thus, there is a significant need for simpler, more efficient Re-ID algorithms suitable for running on multiple edge nodes.

Recent studies have explored distributed frameworks for deep learning applications, particularly using microservices for object detection tasks on edge devices. These frameworks effectively analyze images and videos to extract relevant object information and locations, providing a solid foundation for scalable Re-ID applications involving many cameras or geographic areas [17].

Cloud-based solutions have also been investigated. Video-based person Re-ID using distributed cloud computing stores pedestrian data and model parameters across multiple cloud servers to improve reliability and reduce failures [18]. However, ongoing cloud service costs can be prohibitive for small and medium-sized businesses.

Introducing a message broker into a microservices architecture provides additional advantages. Message brokers enable seamless communication among microservices by handling data exchange, enhancing reliability, and simplifying integration. Specifically, they help Re-ID systems quickly share person-related data across various cameras and processing units, ensuring low latency and better system scalability.

Overall, the shift toward microservices architecture, combined with the use of message brokers, reflects a broader trend towards modular, scalable, and efficient AI system deployment, making it particularly beneficial for distributed person Re-ID solutions in retail and similar settings [19]. To maximize the performance of the system by avoiding bottlenecks in any components of the centralized server, this thesis utilizes a microservices architecture to distribute the computational workload across edge devices and the central server.

2.2 Foundation theory

2.2.1 Object detection

Object detection technology finds applications across numerous domains including automatic traffic violation systems, identification of unfamiliar persons, digital attendance systems, and autonomous robotic vehicles. The advent of deep learning has dramatically enhanced object detection capabilities. Region-Based Convolutional Neural Networks (R-CNN) [20] represented one of the pioneering breakthroughs in this area, combining CNN architectures [21] with region proposal mechanisms to achieve accurate object localization and classification in images. Subsequent iterations, Fast R-CNN [22] and Faster R-CNN [23], were developed to enhance both processing speed and detection precision compared to the original model. Despite these improvements in detection performance, the multi-step processing pipeline made these approaches impractical for real-time applications.

Modern frameworks such as Detectron2 [24] and EfficientDet [25] have pushed object detection forward considerably. Detectron2 offers flexible deployment of high-performing models but demands extensive setup and typically involves computationally heavy architectures, making it unsuitable for real-time or resource-limited environments. EfficientDet provides a more practical option for devices with constrained resources, though it may struggle to meet strict real-time performance criteria.

YOLO model addresses the limitations of multi-stage detection approaches by reformulating object detection as a single regression problem. YOLO processes the entire image in one forward pass, directly predicting bounding boxes and class probabilities from full images. This unified architecture enables real-time performance while maintaining reasonable accuracy for many applications.

The YOLO family has evolved through multiple iterations, with each version improving upon speed-accuracy trade-offs. YOLOv11 [26], in particular, offers several model variants ranging from nano (yolo11n) to extra-large (yolo11x) configurations. The nano variant is specifically designed for resource-constrained environments, featuring significantly reduced parameters and computational requirements while preserving essential detection capabilities.

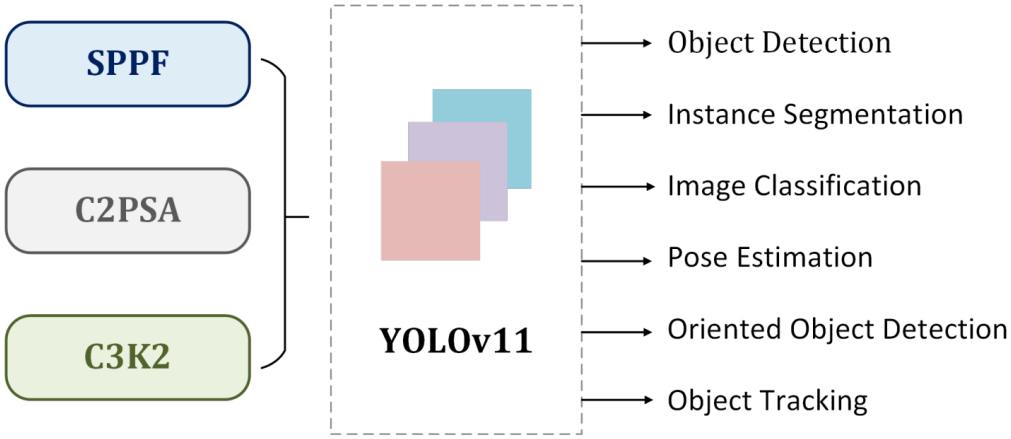

Figure 2.1: Key architectural modules in YOLO11 [26].

A significant advancement in YOLOv11 is the integration of the C2PSA - Convolutional block with Parallel Spatial Attention component, which enhances spatial attention capabilities beyond previous YOLO iterations. The C2PSA block enables the model to focus more effectively on critical regions within images by inaplementing parallel spatial attention mechanisms. This enhancement is particularly beneficial for detecting objects of varying sizes and positions, addressing common challengees in complex visual environments with partially occluded or small objects. The retention of the Spatial Pyramid Pooling - Fast (SPPF) blockfrom previous versions, combined with the new C2PSA component, creates a compreehensive feature processing pipeline that balances computational efficiency with enhancced spatial awareness.

For edge-based human monitoring applications, YOLOv11n provides an optimal balance between detection performance and computational efficiency. Its lightweight architecture enables deployment on edge devices for real-time person detection, serving as the foundation for subsequent tracking and Re-ID processes in distributed

camera networks. For these benefits, YOLOv11n is chosen as the object detection model for this thesis.

2.2.2 Object tracking

Person Re-ID systems face significant challenges when relying solely on frame-by-frame analysis. Individuals frequently lose their visual identity due to various factors including occlusions from other people or objects, rapid movement causing motion blur, and temporary disappearance from camera coverage areas. While deep learning-based feature extraction models can effectively capture contextual information and compute discriminative identity embeddings, frame-based matching approaches often suffer from identity fragmentation where the same person receives multiple different identities across consecutive frames.

To address these limitations, tracking mechanisms play a crucial role in maintaining identity consistency over temporal sequences. Unlike existing methods that perform Re-ID independently for each frame, tracking-based approaches maintain continuous identity associations across time. This temporal continuity significantly outperforms computationally expensive alternatives such as query-driven region proposals [27] and graph-based retrieval methods [28], which become prohibitively costly in large-scale deployment scenarios. By leveraging tracking, our system can efficiently associate multiple detections of the same person, substantially reducing redundant identity searches while improving real-time processing capabilities.

The development of MOT algorithms has evolved through several generations, each addressing specific limitations of previous approaches.

Multi-object tracking has evolved through several approaches, each with distinct trade-offs. SORT [29] provides efficient tracking by integrating object detection with motion prediction, but struggles with complex movement patterns and fast-paced scenarios. To address these limitations, DeepSORT [30] incorporates appearance features via a pre-trained Siamese network, improving performance in dense environments where motion alone is inadequate. However, this enhancement introduces dependency on embedding quality and computational complexity, making it susceptible to visual disturbances. FairMOT [31] advances this paradigm by merging detection and tracking into a unified architecture that simultaneously produces detection outputs and Re-ID features for enhanced multi-object tracking. This unified approach, while effective, demands significant computational resources and requires careful optimization between detection and Re-ID objectives, ultimately compromising processing speed. Alternative solutions include MMTracking [32], which provides a versatile framework supporting multiple advanced algorithms but necessitates

substantial parameter optimization.

ByteTrack [33] represents a breakthrough in tracking methodology, delivering exceptional performance without requiring dedicated appearance models, thereby maintaining high processing speeds particularly in crowded environments. This approach achieves an optimal trade-off between real-time processing and tracking reliability.

Consequently, this thesis employs ByteTrack for maintaining pedestrian identity consistency across video frames, significantly enhancing ID assignment precision. This capability proves essential in dense scenarios where overlapping persons create significant challenges for camera-based identification systems.

2.2.3 Feature extraction

Feature extraction serves as the cornerstone of person Re-ID systems, transforming raw image data into compact descriptors suitable for robust identity matching. To enhance efficiency, especially for deployment on resource-constrained edge devices, recent research has focused on designing specialized lightweight feature extraction architectures.

For instance, Wang et al. introduced the Attention Knowledge-distilled Lightweight Network (ADLN) [34], specifically tailored for edge applications. ADLN utilizes a dimension interaction attention module to improve channel-wise feature representation, complemented by self-distillation that transfers learned attention patterns from deeper layers to shallower ones. Employing a combination of cross-entropy, weighted triplet, and center loss, ADLN effectively minimizes intra-class variability, achieving competitive accuracy on widely-used benchmarks such as Market-1501 and DukeMTMC-ReID, while significantly reducing computational complexity.

Building on similar objectives, Gao et al. presented a joint attention-based Re-ID model [35]. This model emphasizes both global and local pedestrian features through integrated attention modules, achieving precise and discriminative feature extraction that aligns well with realistic surveillance scenarios. This balance between accuracy and computational efficiency makes the model highly suitable for edge deployment.

Recent advances have also explored lightweight Transformer-based architectures, notably LightAMViT [36]. By streamlining self-attention mechanisms through K-means-based token clustering and adaptive weighted pooling, LightAMViT significantly reduces computational demands compared to traditional Vision Transformers. This approach provides a practical alternative, blending the strong representational capability

of Transformers with the computational efficiency required by edge devices.

Complementing these specialized architectures, broader efforts in mobile-optimized methodologies such as MobileNetV2 [37] and MobileNetV3 [38] have gained prominence in edge-based AI applications. These networks use depth-wise separable convolutions to significantly cut down computational overhead while preserving accuracy. Additionally, Squeeze-and-Excitation Networks (SE-Net) [39] introduce adaptive channel-wise recalibration mechanisms, further enhancing model efficiency by focusing computation on informative image regions.

However, despite these general improvements, generic lightweight architectures often lack specific optimizations required by the complexities of person Re-ID tasks. This gap has motivated dedicated research into architectures explicitly designed for Re-ID applications.

Among these specialized designs, OSNet (Omni-Scale Network) [8] represents a significant advancement. With only 2.2 million parameters, OSNet is substantially smaller than traditional methods like ResNet-50, which typically use around 24 million parameters, making it highly suitable for edge deployment. OSNet introduces innovative omni-scale feature learning through a novel building-block design, effectively capturing multi-scale information without the heavy computational cost typically associated with traditional multi-scale approaches. Its use of depth-wise separable convolutions and channel-shuffling operations maintains computational efficiency while preserving strong discriminative capability.

Expanding on OSNet's approach, LightMBN (Lightweight Multi-Branch Network) [9] further advances edge-friendly Re-ID through a multi-branch design tailored for constrained environments. LightMBN integrates efficient channel-wise and spatial attention mechanisms, adaptively emphasizing informative features without significantly increasing computational load. This enables robust identity matching with minimal resource usage.

Given these advantages, this thesis adopts OSNet and LightMBN as the feature extraction models for the proposed system, ensuring lightweight but still enough efficient for person Re-ID processing.

2.2.4 Image classification

Traditionally, image classification and person Re-ID have been considered distinct tasks, with limited overlap. Person Re-ID focuses on matching individuals across different camera views, while image classification typically identifies broad object categories. However, when considering scenarios with a large search space-potentially

thousands of identities - performing direct identity matching can be computationally expensive and slow. Under these conditions, reducing the search space using easily classifiable human metadata, such as gender, becomes beneficial by limiting the number of candidates considered, thus improving retrieval speed and accuracy.

Commonly used image classification models include well-established convolutional neural networks (CNNs) such as ResNet [40], known for its deep residual learning framework that significantly improves accuracy in classification tasks. MobileNet [41] offers lightweight architectures optimized for resource-constrained devices, leveraging depth-wise separable convolutions to reduce computational costs. More recently, Vision Transformers (ViT) [42] have introduced transformer-based architectures to image classification, leveraging self-attention mechanisms to capture global relationships, achieving impressive accuracy at the cost of higher computational demands.

To effectively balance accuracy and computational efficiency in edge-based systems, EfficientNet [43] was introduced. EfficientNet leverages a compound scaling method to optimally adjust network depth, width, and resolution. Among its variants, EfficientNet-B0 stands out due to its particularly compact structure and excellent performance, making it ideal for deployment on lightweight edge devices.

By employing EfficientNet-B0 for gender classification in the proposed Re-ID pipeline, the system effectively filter candidate matches by gender, significantly reducing computational overhead and improving the efficiency of subsequent identity matching stages.

2.2.5 Message queue

In hybrid edge-server deployments for person Re-ID systems, efficiently handling data flow between distributed devices is crucial. A reliable message queue system helps manage communication among edge devices, such as cameras and IoT sensors, and central processing servers. Message queues facilitate asynchronous and stable data transfer, allowing edge devices to process data locally without delays caused by direct server interactions.

Several widely-used message queuing solutions exist, such as Apache Kafka, RabbitMQ, and ZeroMQ, each with unique strengths:

RabbitMQ [44] is a well-established messaging broker that supports complex routing patterns and guaranteed message delivery. It is particularly effective for transactional systems where reliable delivery and message acknowledgment are vital. However, RabbitMQ can face scalability challenges when managing the high-

throughput streaming data common in video-based person Re-ID scenarios.

ZeroMQ [45] is lightweight, extremely fast, and suitable for direct, point-to-point messaging. Its simplicity and speed make it ideal for real-time data transmission, but it lacks built-in persistence and fault tolerance mechanisms necessary for hybrid edge-server deployments that require robust data management over unreliable networks.

Considering these aspects, *Apache Kafka* [46] emerges as the most suitable choice for hybrid edge-server person Re-ID systems. Kafka excels in handling real-time streaming data, providing strong scalability, reliability, and fault-tolerant message persistence. It efficiently processes high volumes of continuous data, such as features extracted from surveillance video streams, ensuring seamless integration between multiple distributed edge nodes and centralized processing servers.

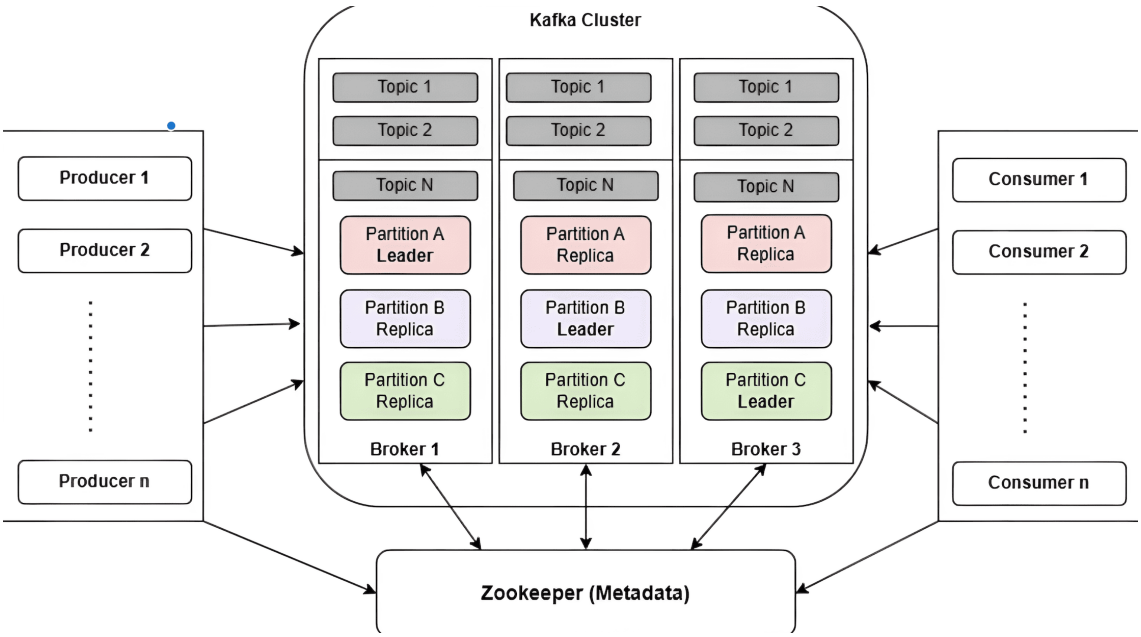

Figure 2.2: Kafka Architecture illustrating Producers, Consumers, Topics, Partitions, and Zookeeper [47].

In Kafka's architecture, *producers* generate data streams-such as encoded video frames in byte format, video metadata, and detected human bounding boxes-and send these streams to Kafka's storage system. In the context of person Re-ID, producers correspond to edge devices (e.g., surveillance cameras or edge processors). On the other hand, *consumers* are processes initiated on central servers, which simultaneously subscribe to these data streams to retrieve and perform further processing, such as feature extraction, identity matching, and analytics. Kafka organizes these data streams into logical channels known as *topics*, each representing a specific category or type of data. To enhance scalability, each topic is divided into multiple *partitions*, enabling parallel processing and improved fault tolerance across distributed

environments.

This structured approach ensures Kafka can effectively manage data flow and scale seamlessly across distributed nodes. For person Re-ID pipelines specifically, Kafka's ability to handle high-throughput video streams, maintain message ordering within partitions, and provide durable storage makes it exceptionally well-suited for scenarios where edge devices continuously generate feature vectors and metadata that must be reliably transmitted to central servers for identity matching. Furthermore, Kafka's built-in replication and fault-tolerance mechanisms ensure data integrity even when individual edge nodes experience connectivity issues, while its horizontal scaling capabilities allow the system to accommodate growing numbers of surveillance cameras and processing nodes without performance degradation.

These characteristics make Kafka not just suitable, but optimal for complex, distributed Re-ID deployments where reliability, scalability, and real-time data processing are essential.

2.2.6 Model serving framework

Deploying person Re-ID and lightweight classification models effectively in hybrid edge–server architectures requires selecting an appropriate model-serving framework. Several well-known solutions such as TorchServe, TensorFlow Serving, and NVIDIA Triton Inference Server have been widely adopted. TorchServe [48] is popular due to its native support and ease of use with PyTorch models, featuring straightforward REST APIs and dynamic batching to enhance GPU throughput. However, it lacks the ability to concurrently serve multiple instances of the same model efficiently on a single GPU, limiting maximal GPU utilization for lightweight models. TensorFlow Serving [49], while efficient in serving TensorFlow models, presents additional complexity for PyTorch-based workflows due to the necessity of model conversion. NVIDIA Triton Inference Server [50], meanwhile, excels at maximizing GPU usage through concurrent model execution and dynamic batching, enabling high throughput and efficient GPU resource allocation. Nevertheless, Triton's complexity and configuration overhead can present challenges, particularly for development teams prioritizing rapid deployment and ease of integration.

To balance these trade-offs, Ray Serve [51] has emerged as an attractive framework, offering notable flexibility and ease of integration. Specifically designed for scalable deployments, Ray Serve supports serving multiple models simultaneously, dynamically composing inference pipelines, and efficiently managing replicas to maximize GPU utilization. Additionally, it seamlessly integrates with FastAPI, allowing developers to directly embed their inference endpoints within Python-based microservice architectures.

This combination of flexible model orchestration, autoscaling capabilities, and developer-friendly integration makes Ray Serve particularly suited for deploying lightweight person Re-ID models like OSNet and LightMBN, as well as efficient image classification models, within hybrid edge–server setups.

Considering these advantages, this thesis adopts Ray Serve as the model serving framework for the proposed system, ensuring efficient deployment of lightweight models in hybrid edge-server architectures to maximize the utilization of hardware resources.

2.2.7 Containerization

Containerization is a foundational technology in modern DevOps and microservices architectures, enabling applications and their dependencies to be packaged into lightweight, isolated units. This approach simplifies development workflows and significantly improves service portability. By encapsulating services within containers, they can be deployed, scaled, and updated independently, which facilitates more efficient version control and lifecycle management in distributed systems.

Among the various container technologies, Docker [52] is a widely used tool for building and running containers. Docker Engine serves as the core runtime that manages container lifecycle, including creation, execution, and resource allocation. Developers define application environments in a ‘Dockerfile‘, which contains step-by-step instructions for building container images, including base operating system, dependencies, configuration files, and application code. This declarative approach ensures consistent behavior across both staging and production environments, eliminating the common "it works on my machine" problem. The Docker Engine utilizes Linux kernel features such as namespaces for process isolation and cgroups for resource management, enabling multiple containers to run securely on a single host without interference. Additionally, Docker’s layered filesystem architecture allows for efficient image storage and sharing, where common layers are reused across different containers, reducing storage overhead and improving deployment speed.

Overall, with Docker, the system can be deployed on any machine with Docker installed, ensuring consistent behavior across different environments. Given these advantages, this thesis adopts Docker as the containerization technology using for both edge device and centralized server deployment.

2.2.8 Vector database

Traditional databases, such as OLTP (Online Transaction Processing) and OLAP (Online Analytical Processing) systems, are optimized for structured queries and aggregated data operations. However, these systems typically face limitations when

handling high-dimensional, continuous data like image feature embeddings. In contrast, **vector databases** are specifically designed to efficiently store and query vector representations, enabling rapid similarity searches based on distance metrics, notably cosine similarity.

Although widely used in Natural Language Processing (NLP) tasks like Retrieval-Augmented Generation (RAG), vector databases also significantly contribute to the person Re-ID domain by facilitating efficient identity matching. In person Re-ID systems, each individual's identity is stored as a high-dimensional feature vector-commonly 512 or 1024 dimensions-depending on the chosen deep learning feature extractor. During the retrieval stage, the system queries this vector database with a new feature vector to identify the most similar stored identities.

Several popular vector databases have emerged, each with distinct characteristics:

- *Faiss*, developed by Meta, is a highly efficient C++ library offering both exact and approximate nearest-neighbor search using indexing strategies like Hierarchical Navigable Small World (HNSW), product quantization, and GPU acceleration [53].
- *Milvus* is a scalable, distributed vector database built upon Faiss and hnswlib, supporting vast volumes of vectors and hybrid indexing methods optimized for distributed deployments [54].
- *ChromaDB* is lightweight and optimized for NLP embeddings, providing simplicity and ease of use, though it may not offer the highest performance for large-scale or complex metadata queries [55].
- *Qdrant*, a Rust-based vector database, excels in hybrid queries due to its built-in support for efficient metadata filtering combined directly with vector search. Its HNSW indexing structure integrates metadata filtering during the search, significantly reducing query latency compared to post-filtering approaches [56].
- *Redis*, traditionally a key-value store, now includes vector search capabilities through the RediSearch module. Redis offers exceptional speed and convenience with its in-memory architecture, providing low-latency vector similarity searches while maintaining familiar Redis operations for metadata management. Its mature ecosystem and widespread adoption make it particularly attractive for hybrid edge-server deployments [57].

Vector database indexes, such as HNSW [58], differ fundamentally from traditional database indexes. Rather than focusing on exact matches or sorted queries, they organize vectors into graph-based structures optimized for approximate nearest-

neighbor searches in high-dimensional spaces. The most commonly used distance metric in such scenarios is cosine similarity, which measures the angular similarity between vectors.

Among these alternatives, *Redis* emerges as the most suitable choice for our hybrid edge-server person Re-ID pipeline. Its in-memory architecture provides exceptional query performance, while its mature ecosystem and widespread adoption ensure reliable deployment and maintenance. Although Redis may have more limited advanced metadata filtering capabilities compared to specialized vector databases like Qdrant, its simplicity, speed, and integration convenience make it ideal for real-time person Re-ID applications where low latency is critical. Additionally, Redis's ability to handle both vector similarity searches and traditional key-value operations within a single system simplifies the overall architecture and reduces operational complexity.

CHAPTER 3. METHODOLOGY

Building upon the theoretical foundations established in Chapter 2, this chapter presents a comprehensive examination of the proposed system's design and implementation. It details (i) the overall hybrid edge-server architecture, (ii) the specific hardware and software strategies for resource-constrained edge devices, (iii) the architecture of the centralized server, including its use of microservices for intensive AI tasks, (iv) the crucial role of the Apache Kafka cluster as the communication backbone, and (v) a novel optimization technique that uses person metadata (gender) to enhance identity retrieval efficiency.

3.1 Overview

This research presents a comprehensive, scalable person Re-ID system that utilizes the capabilities of distributed edge devices. The proposed architecture consists of three primary components: edge computing devices, a central processing server, and a message broker cluster implemented with Apache Kafka. Each has distinct roles to efficiently distribute tasks and optimize performance.

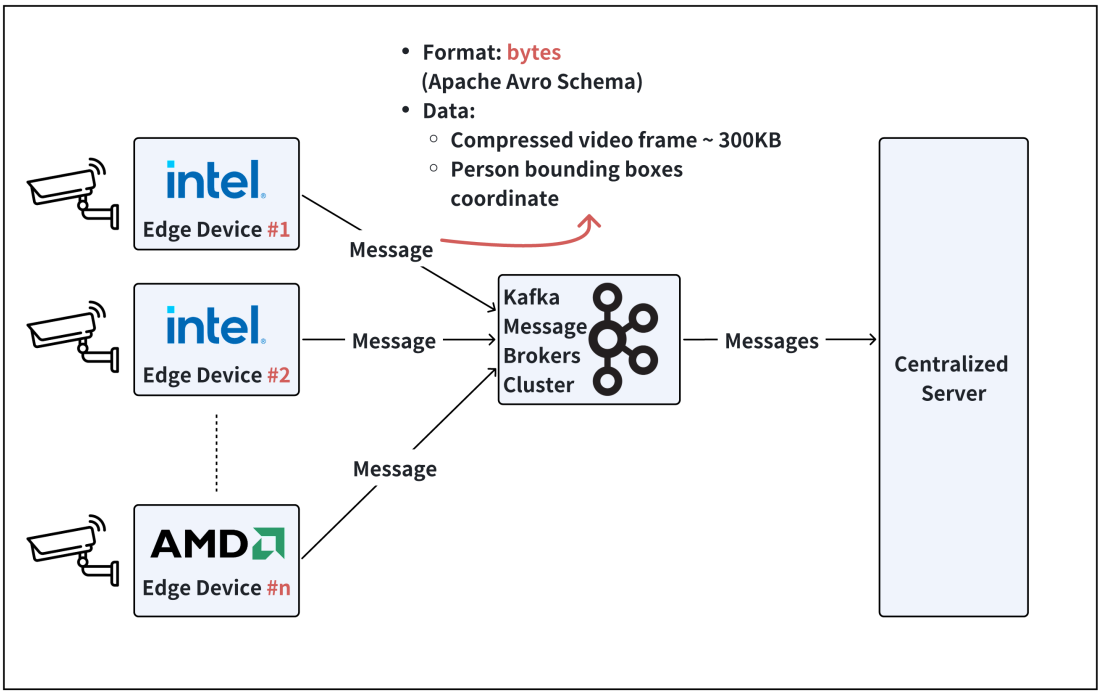

Figure 3.1: System overview of the hybrid edge-server person Re-ID pipeline showing the distributed setup with edge devices doing human detection, Kafka message broker cluster asynchronously handling communication, and central server managing model inference support for identity matching across multiple cameras.

- **Edge devices (Producers)**: Each edge device is responsible for capturing and preprocessing video streams from surveillance cameras. Given the limited

resources typically available at the edge-specifically, a CPU-only setup with minimal specifications (1 CPU at 3.5 GHz and 1 GB RAM, roughly equivalent to 7.5 GFLOPS) using $\times 86$ architecture (Intel or AMD processors)-the device focuses solely on performing essential tasks like human detection and frame compression. Notably, no GPU acceleration is required here since these initial processing tasks, particularly when using highly optimized models, can be executed efficiently on a CPU. To meet the constraints of such limited hardware, we utilize YOLOv11n, the smallest and most compact version of the YOLOv11 model family, ensuring acceptable inference speed and latency without specialized accelerators. This approach enables reliable real-time processing at the edge while conserving resources for more demanding tasks on the server side.

- **Kafka message brokers cluster**: A cluster of three Kafka brokers ensures high availability, fault tolerance, and reliable message management. Each edge device sends messages asynchronously to this Kafka cluster, packaging compressed video frames (approximately 300 KB each) and bounding box coordinates using a structured format defined by Apache Avro schemas. Kafka efficiently handles message buffering and transmission, preventing data loss even during temporary network disruptions. This Kafka cluster is deployed as a containerized service using Docker, simplifying management, scalability, and deployment across the hybrid infrastructure.
- **Centralized server (Consumers)**: The centralized server manages several internal services critical to the Re-ID pipeline, including model serving frameworks, vector databases, and tracking modules. Equipped with a robust GPU, this server is optimized for computationally intensive tasks such as feature extraction, classification (e.g., gender), and vector similarity searches for identity matching. Multiple Kafka consumers are deployed, with each consumer dedicated to processing data streams from a specific edge device (camera). By interacting with a shared vector database, these consumers enable accurate cross-camera identity matching. This design allows effective parallel processing of multiple video streams, significantly enhancing scalability and real-time performance in complex multi-camera environments.

This division of labor among the components ensures optimal resource utilization and robust system performance, delivering efficient, accurate, and scalable person Re-ID in real-world deployments.

3.2 Kafka message brokers cluster

As illustrated in the system overview (Figure 3.1), the Apache Kafka cluster is crucial to the system architecture, serving as a real-time messaging backbone between CPU-based edge devices and the centralized server. Kafka ensures efficient, reliable, and scalable data streaming, effectively mitigating several potential critical issues inherent to direct edge-server communication:

- **Bottleneck**: A bottleneck refers to a point in a system that restricts overall performance due to limited throughput capabilities. In practical scenarios, each edge device streams data at an average rate of 12-20 FPS. Thus, with just 10 devices, the centralized server receives approximately 120-200 messages per second. This volume of messages can quickly exceed the server's processing capacity, even when supported by powerful hardware.

  Additionally, network bandwidth becomes a significant constraint. For instance, at 12 FPS with each message roughly 1 MB in size (including FullHD video frames, bounding box coordinates, and metadata such as frame ID and timestamps), the server processes about 12 MB/s per device. Consequently, a system of 10 devices generates approximately 120 MB/s (0.96 Gbps), necessitating extremely high and stable bandwidth to prevent congestion.

  Kafka alleviates these bottlenecks by buffering and efficiently distributing message loads, allowing smooth and reliable server processing.

- **Loss of Data**: Without Kafka, edge devices would directly transmit data to the centralized server, posing risks of significant data loss due to network disruptions (such as weather interference or intermittent connectivity). Kafka's design inherently provides message durability through its distributed log structure, ensuring that messages are stored persistently until successfully processed, thus greatly reducing potential data loss.

- **Synchronization**: Direct edge-server communication would necessitate complex synchronization mechanisms to maintain the correct order of frames, a critical requirement for accurate tracking in Re-ID applications. Kafka inherently ensures ordered message processing by managing partitioned logs effectively. Thus, it simplifies synchronization significantly, guaranteeing the sequential integrity of frames from each camera source.

Due to these critical considerations, Apache Kafka plays an indispensable role, providing robust, efficient, and reliable data management essential for the seamless operation of the Re-ID pipeline.

3.2.1 Cluster configuration

A Kafka cluster is a collection of multiple Kafka brokers that work together, managed by a single Zookeeper node or a cluster of Zookeeper nodes. Based on the properties of the current Re-ID system, a cluster includes three brokers and one Zookeeper node is selected to ensure high availability and fault tolerance, as illustrated in Figure 3.2.

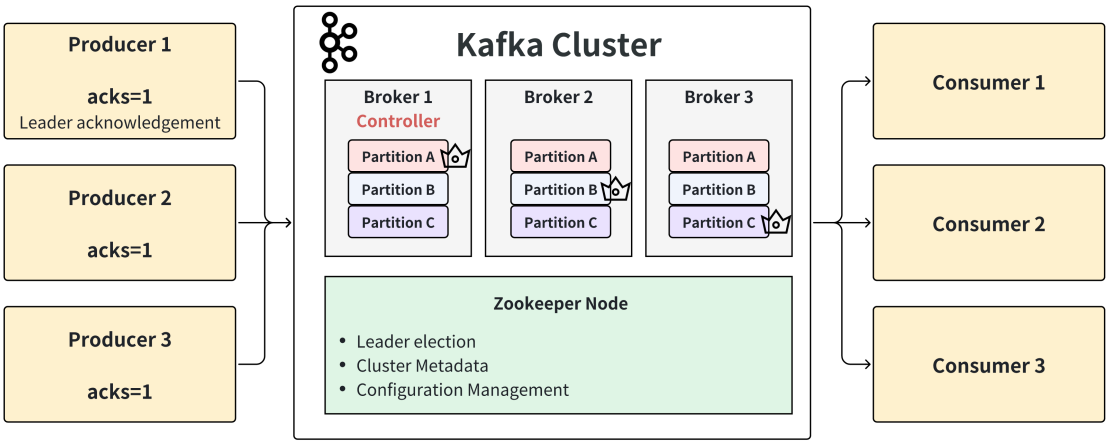

Figure 3.2: Kafka clusters with three brokers and one Zookeeper node.

Zookeeper node The Zookeeper service manages cluster metadata, including broker registration, topic configuration, partition leadership information, and consumer group coordination. It ensures consistent configuration management across the entire Kafka ecosystem.

Kafka brokers The three Kafka brokers (Broker 1, Broker 2, and Broker 3) form the core of the Kafka cluster, as depicted in Figure 3.2. These brokers are responsible for storing data, managing message partitions, and enabling communication between producers and consumers. Each broker hosts different partitions of the data topics:

- *Controller*: One broker acts as the cluster controller. The controller handles administrative tasks such as assigning partitions to brokers, managing leadership elections for partitions, and monitoring the overall cluster state.
- *Topics*: Topics represent logical streams of records categorized by specific names, serving as the fundamental organizational unit for data flow within Kafka. While topics are conceptual abstractions rather than physical entities, they provide essential structure for message categorization and routing. In the context of this Re-ID system, topics such as `reid_input` are used to channel

video frame data and metadata from edge devices to the centralized server. Each topic can accommodate multiple producers (edge devices) simultaneously writing data and multiple consumers (server-side processing modules) reading data, enabling flexible many-to-many communication patterns essential for scalable distributed processing.

- *Partitions*: Each topic is divided into partitions (based on initial setup) to facilitate scalability and parallelism. In Figure 3.2, these partitions are distributed across the three brokers, with each partition having a designated leader (marked by a crown icon). The leader partition is responsible for managing read and write requests from producers and consumers. Follower partitions serve as replicas to ensure data redundancy and fault tolerance.

Producers and Consumers The system includes multiple producers representing the edge devices that send messages to the Kafka cluster with an acknowledgment setting of `acks=1`, meaning producers wait for confirmation only from the leader broker before proceeding. On the consumer side, multiple consumers subscribe to Kafka topics to read and process data. Messages from partitions are delivered to consumers based on their subscription configuration and current offset position.

The coordination of these distributed producers and consumers, along with the management of broker leadership and partition assignments, requires a centralized coordination service. This critical orchestration role is fulfilled by Apache ZooKeeper, which serves as the backbone for maintaining cluster metadata and ensuring consistent distributed state management across all Kafka components.

To comprehensively understand the operational mechanics of the Kafka cluster within this Re-ID system, it is essential to examine the following fundamental concepts and their specific implementations:

3.2.2 Zookeeper node

In this Re-ID system, we use a single Zookeeper node instead of the common 3-node ensemble. This choice is based on the system's needs and practical reasons:

- **Resource Efficiency**: For the current scale (3 Kafka brokers with moderate message flow), one Zookeeper node is enough to manage coordination without wasting extra computing power or network resources that could be used by the Re-ID system itself.
- **Simpler Management**: Having a single node reduces complexity, especially for small organizations or businesses that may not have dedicated DevOps staff. It also makes monitoring, backups, and troubleshooting easier.

- **Development and Testing**: This setup works well for development, testing, and early production stages where quick deployment and cost saving are more important than full availability.

For larger or more critical deployments, it is easy to switch to a 3-node Zookeeper ensemble to improve reliability.

Partition Assignment and Producer Acknowledgement As configured, the producer's acknowledgment setting is `acks = 1`, which means the producer waits for a confirmation only from the leader partition before considering the message successfully sent. This setting provides a balance between latency and data safety, allowing faster message delivery while ensuring that the leader has safely written the message.

For partition assignment, Kafka uses a round-robin strategy when no specific key is provided. For the `reid_input` topic with 6 partitions, the partition ID for the i-th message is determined by:

$$\text{Partition ID} = i \bmod 6, \quad i = 0, 1, 2, \ldots \tag{3.1}$$

This ensures messages are evenly distributed across all partitions in a cyclic manner:

$$0, 1, 2, 3, 4, 5, 0, 1, 2, \ldots$$

Thus, with `acks = 1`, the producer receives acknowledgment as soon as the leader partition confirms the message write, optimizing throughput while maintaining reasonable reliability.

Leader Partition Election Kafka's leader election mechanism ensures high availability and fault tolerance through the following process:

1. **Initial Leader Assignment**: When a topic is created, Kafka automatically assigns one replica as the leader for each partition. The controller broker (Broker 1 in this configuration) manages this initial assignment based on the replica placement strategy.
2. **In-Sync Replica (ISR) Maintenance**: The leader maintains a list of In-Sync Replicas that are fully caught up with the leader's log. Only ISR members are eligible for leadership.
3. **Leader Failure Detection**: Zookeeper monitors broker health through heartbeat

mechanisms. When a leader becomes unavailable, the controller broker detects this failure.

4. **New Leader Selection**: The controller selects a new leader from the ISR list, typically choosing the first available replica in the preferred replica list to maintain optimal load distribution.
5. **Metadata Update**: Once a new leader is elected, the controller updates the cluster metadata and notifies all brokers and clients about the leadership change.

This election process typically completes within seconds, ensuring minimal disruption to the Re-ID data processing pipeline.

3.2.3 Kafka brokers

The Kafka message broker cluster serves as the communication backbone between edge devices and the centralized server in the distributed Re-ID system. This section details the broker configuration, message serialization strategies, and the rationale behind our architectural decisions.

a, Broker configuration and topology

The proposed Kafka cluster employs a three-broker architecture deployed within a containerized environment using Docker, strategically designed to achieve optimal fault tolerance while maintaining resource efficiency for deploying on resource-constrained devices. This distributed topology addresses the critical requirements of real-time person Re-ID systems through carefully orchestrated broker configurations and message flow patterns.

The cluster architecture incorporates the following key specifications:

- **Broker Count**: Three strategically distributed brokers (Broker 1, Broker 2, Broker 3) provide high availability through redundancy while maintaining cost-effectiveness. This configuration enables the system to continue operating even with single-broker failures, ensuring uninterrupted Re-ID processing. The broker count can be dynamically scaled based on edge device proliferation and processing demands.
- **Replication Factor**: Critical topics maintain a replication factor of 3, ensuring complete data redundancy across all brokers. This configuration guarantees zero data loss during broker failures and enables seamless failover mechanisms for continuous operation.
- **Topic Partitioning**: The primary `reid_input` topic is strategically partitioned into 6 segments, enabling parallel processing capabilities and intelligent load

distribution across multiple consumer instances. This partitioning strategy maximizes throughput while maintaining message ordering within individual partitions.

- **Containerization**: Each broker operates within isolated Docker containers, providing deployment flexibility, horizontal scaling capabilities, and simplified maintenance operations. This containerized approach facilitates rapid deployment across diverse infrastructure environments.

The broker configuration is specifically optimized for video data transmission characteristics inherent to Re-ID applications. Each broker maintains sufficient memory allocation to handle the buffering of compressed video frames, accommodating message sizes ranging from 200-400 KB depending on scene complexity, motion dynamics, and compression algorithms. The configuration prioritizes both message throughput and reliability, incorporating adaptive batching mechanisms and optimized serialization protocols to minimize latency while ensuring data integrity throughout the Re-ID processing pipeline.

b, Message serialization

In distributed Re-ID systems where edge devices communicate with central servers through Apache Kafka, a critical challenge emerges: efficiently encoding messages that contain both structured metadata and binary image data. The fundamental issue stems from the need to serialize heterogeneous data types-including device identifiers, detection results, timestamps, and raw image frames-into a single message format suitable for network transmission.

The primary constraint lies in the incompatibility between text-based serialization formats and binary data representation. Traditional approaches require careful consideration of encoding overhead, transmission efficiency, and system scalability.

Baseline Approach: Base64 Encoding with JSON

The conventional solution involves converting binary image data to Base64 encoding before embedding it within a JSON structure. This approach follows a four-step process:

1. **Binary-to-Text Conversion**: Raw image data is encoded using Base64 algorithm to produce ASCII string representation
2. **JSON Structure Formation**: The Base64 string is embedded as a field value within the JSON message structure containing metadata
3. **Serialization**: The complete JSON object is serialized to bytes for network transmission

4. **Kafka Transmission**: The encoded message is sent through Kafka topics

The JSON message structure with base64 encoded image data follows this format:

```
{
    "device_id": "camera_001",
    "frame_number": 12345,
    "image_data": "iVBORw0KGgoAAAANSUhEUgAA...",
    "created_at": 1640995200000,
    "result": [...]
}
```

Figure 3.3: JSON message structure with Base64 encoded image data.

This encoding is necessary because JSON, being a text-based format, cannot natively represent binary data. JSON supports only primitive types (string, number, boolean) and structural types (object, array), making text encoding the only viable option for binary data inclusion.

However, this approach introduces significant overhead. For a typical Full HD image frame with an original size of 2MB, the Base64 encoding process increases the message size to approximately 3MB - a 50% increase. This expansion occurs due to Base64's inherent 33% size inflation factor, compounded by JSON structural overhead.

To address the limitations of text-based encoding, we propose using Apache Avro as the primary serialization framework. Avro can be conceptualized as a binary equivalent of JSON that natively supports raw binary data without requiring text encoding, and also provide schema validation capabilities.

Key Advantages:

- **Native Binary Support**: Direct handling of binary data without Base64 conversion
- **Apache Ecosystem Compatibility**: Seamless integration with Kafka, Schema Registry, and related tools
- **Schema Evolution**: Built-in support for backward and forward compatibility
- **Cross-Language Support**: Language-agnostic serialization format

The Avro schema definition for our edge device messages is structured as follows:

```
{
  "type": "record",
  "name": "EdgeDeviceMessage",
  "namespace": "com.edge.device",
  "fields": [
    {"name": "device_id", "type": "string"},
    {"name": "frame_number", "type": "long"},
    {"name": "result", "type": {
      "type": "array",
      "items": {
        "type": "record",
        "name": "Detection",
        "fields": [
          {"name": "bbox", "type": {
            "type": "array", "items": "float"}},
          {"name": "confidence", "type": "float"},
          {"name": "class_id", "type": "int"}
        ]
      }
    }},
    {"name": "created_at", "type": "long"},
    {"name": "image_data", "type": "bytes"}
  ]
}
```

Figure 3.4: Avro schema definition for edge device messages containing video frames and detection metadata.

The schema defines in Figure 3.4 a structured message format called `EdgeDeviceMessage` within the `com.edge.device` namespace. Each message contains:

- **device_id** (string): Unique identifier for the edge device/camera sending the message
- **frame_number** (long): Sequential frame counter for maintaining temporal order and detecting dropped frames
- **result** (array): Collection of detection results, where each detection includes:
  - **bbox** (array of floats): Bounding box coordinates in normalized format [x1, y1, x2, y2]
  - **confidence** (float): Detection confidence score (0.0 to 1.0) indicating the model's certainty
  - **class_id** (int): Object class identifier (typically 0 for person in person detection models)

- **created_at** (long): Unix timestamp marking when the frame was processed on the edge device
- **image_data** (bytes): Raw binary image data (typically JPEG-compressed) avoiding Base64 encoding overhead

This schema structure enables efficient serialization of both structured metadata and binary image data while maintaining compatibility with Kafka's distributed messaging system. The use of native binary support for `image_data` eliminates the 33% size overhead associated with Base64 encoding, significantly reducing network bandwidth requirements for high-resolution video streams.

c, Topic settings

In this project, two topics are used to facilitate the communication between edge devices and the centralized server:

- `reid_input`: The topic for edge devices to send messages to the Kafka cluster. On server side, each consumer will be assigned to a different consumer group to process different partitions of the topic, hence ensure that each consumer group will only fetch data from one partition (one camera).

  Below code shows the topic settings for `reid_input`:

```
number_of_partitions: 6
replication_factor: 3
min_insync_replicas: 2
cleanup_policy: delete
retention_time: 1 day
max_size_on_disk: 20 GB
max_message_size: 20 MB
```

  The `replication_factor = 3` and `min_insync_replicas = 2` configuration ensures robust fault tolerance and high availability for the Re-ID system. With a replication factor of 3, each partition maintains three copies (one leader and two followers) distributed across the three Kafka brokers. This configuration can tolerate the failure of one broker without data loss or service interruption.

  For example, consider Partition 0 of the `reid_input` topic:

  – **Leader**: Broker 1 (handles all read/write operations)

  – **Follower 1**: Broker 2 (maintains synchronized replica)

  - **Follower 2**: Broker 3 (maintains synchronized replica)

  The `min_insync_replicas = 2` setting requires that at least 2 replicas (leader + 1 follower) must acknowledge each write operation before considering it successful. This ensures that even if one broker fails, the data remains available and consistent. For instance:

  Normal Operation: All 3 replicas are in-sync, writes are acknowledged by leader + 1 follower.

  Single Broker Failure: If Broker 3 fails, Partition 0 still operates normally with Broker 1 (leader) and Broker 2 (follower) maintaining the minimum 2 in-sync replicas.

  Critical Failure Scenario: If 2 brokers fail simultaneously, the partition becomes read-only to prevent data inconsistency, ensuring data integrity over availability.

- `reid_output`: The topic for storing processed frames with comprehensive metadata including person identifications, bounding boxes, and descriptive text annotations. This enables the client-side web application to interactively buffer and visualize real-time Re-ID results for users, providing a seamless interface for monitoring person tracking across the camera network.

```
number_of_partitions: 3
replication_factor: 3
min_insync_replicas: 2
cleanup_policy: delete
retention_time: 6 hours
max_size_on_disk: 5 GB
max_message_size: 10 MB
```

  The `reid_output` topic is configured with fewer partitions (three and six respectively) since output messages are typically aggregated results with lower frequency compared to continuous video frame input. The reduced retention time (six hours vs one day) reflects the time-sensitive nature of Re-ID results, where older identification data becomes less relevant for real-time tracking applications. The smaller maximum message size (10 MB vs 20 MB) accommodates processed metadata and identity vectors rather than raw video frames.

3.2.4 Producers

The Kafka producer configuration is essential for achieving efficient and reliable message delivery from edge devices to the Kafka cluster. Configuration parameters

are optimized for the Re-ID system's real-time streaming requirements, balancing performance, data integrity, and system reliability.

```
self.producer = KafkaProducer(
    client_id=self.device_id,
    bootstrap_servers=self.kafka_bootstrap_servers,
    key_serializer=lambda x: x.encode("utf-8"),   # Edge ↩
↪ device ID
    value_serializer=self.serialize_message,      # ↩
↪ Serialize the message using Avro
    linger_ms=10,
    batch_size=16384,
    acks=1,                                       # 0: No ↩
↪ ack, 1: Leader ack, 'all': All replicas ack
    max_in_flight_requests_per_connection=5,
)
```

Figure 3.5: Kafka producer configuration for edge devices.

The producer configuration parameters are strategically chosen to balance throughput, latency, and reliability for the Re-ID streaming pipeline:

- *client_id*: Uniquely identifies each edge device producer within the Kafka cluster using the device ID. This facilitates monitoring, debugging, and tracking message flow from specific devices through the Kafka UI and broker logs.
- *bootstrap_servers*: Specifies the initial list of Kafka broker addresses for establishing cluster connectivity. The producer uses this list to discover the full cluster topology and partition leadership information.
- *key_serializer*: Serializes the message key (device ID) as UTF-8 encoded strings. The key serves dual purposes: ensuring message ordering per device and enabling consistent partition assignment for load balancing across the cluster.
- *value_serializer*: Employs a custom Avro-based serialization function that efficiently encodes both structured metadata and binary image data without the overhead of text-based encoding schemes like Base64.
- *linger_ms = 10*: Introduces a minimal 10-millisecond delay before sending batches, allowing the producer to collect multiple messages for batch transmission. This significantly improves network utilization and throughput while maintaining sub-frame latency for real-time processing.
- *batch_size = 16384 bytes (16KB)*: Sets the maximum batch size for grouping messages before transmission. This value is optimized for the typical message

size in our Re-ID system, where each message contains compressed image data and detection results.

- *acks = 1*: Configures acknowledgment requirements to wait only for the partition leader's confirmation before considering a message successfully sent. This provides an optimal balance between delivery guarantees and latency, suitable for near real-time applications where occasional message loss is acceptable.
- *max_in_flight_requests_per_connection = 5*: Allows up to 5 unacknowledged requests per broker connection, enabling pipeline parallelism while preserving message ordering within each partition. This setting optimizes throughput without compromising the sequential nature of video frame processing.

This configuration ensures optimal performance for the Re-ID system's streaming requirements, achieving high throughput for continuous video frame processing while maintaining low latency for real-time person tracking and identification tasks.

3.2.5 Consumers

Similar to the producer configuration, the Kafka consumer setup requires common connectivity parameters. However, consumers also need additional specialized settings tailored explicitly for efficiently managing high-volume video streams and computationally demanding Re-ID processing tasks on the centralized server.

The consumer configuration emphasizes parameters specific to server-side processing and high-volume data consumption:

- *auto_offset_reset = "earliest"*: Critical for Re-ID continuity-new consumer instances begin from the earliest available messages, ensuring no video frames are missed during consumer restarts or scaling operations.
- *enable_auto_commit = True*: Simplifies offset management for the idempotent Re-ID processing pipeline where occasional message reprocessing is acceptable compared to the complexity of manual offset management.
- *group_id*: Enables horizontal scaling through consumer groups, allowing multiple server instances to process different partitions simultaneously while maintaining load balance across the Re-ID pipeline.
- *session_timeout_ms = 30000*: Extended 30-second timeout accommodates GPU-intensive operations (feature extraction, similarity searches) that may cause temporary processing delays, preventing unnecessary consumer group rebalancing.
- *heartbeat_interval_ms = 3000*: Maintains responsive group membership while

```
self.consumer = KafkaConsumer(
    self.input_topic_name,
    client_id=self.client_id,
    bootstrap_servers=self.kafka_bootstrap_servers,
    auto_offset_reset="earliest",
    enable_auto_commit=True,
    group_id=self.consumer_group,
    session_timeout_ms=30000,                    # 30 seconds
    heartbeat_interval_ms=3000,
    max_poll_interval_ms=300000,                 # 5 minutes
    retry_backoff_ms=100,                        # Time to ↩
  ↪ wait before retrying
    reconnect_backoff_ms=1000,                   # Time to ↩
  ↪ wait before reconnecting
    reconnect_backoff_max_ms=1000,               # Maximum ↩
  ↪ time to wait before reconnecting
    fetch_min_bytes=1024 * 1024,                 # 1MB
    fetch_max_bytes=100 * 1024 * 1024,           # 100MB
    max_partition_fetch_bytes=100 * 1024 * 1024,
)
```

Figure 3.6: Kafka consumer configuration for centralized server.

staying well below the session timeout threshold, ensuring timely failure detection without overwhelming the cluster with heartbeat traffic.

- *max_poll_interval_ms = 300000*: Allows 5-minute processing windows for complex Re-ID operations including vector database queries, similarity computations, and identity updates, preventing consumer eviction during intensive processing phases.
- *Fetch Configuration (fetch_min_bytes, fetch_max_bytes, max_partition_fetch_bytes)*: Optimized for large video messages-minimum 1MB accumulation improves network efficiency while maximum 100MB limits prevent memory overflow. The increased partition fetch size (from default 1MB to 100MB) eliminates fetch size bottlenecks for HD video frames with detection metadata.
- *Reconnection Strategy (retry_backoff_ms, reconnect_backoff_max_ms)*: Aggressive reconnection with minimal delays ensures rapid recovery for time-sensitive Re-ID processing where extended disconnections significantly impact tracking continuity.

This server-side configuration complements the edge device producer settings,

creating an end-to-end optimized messaging pipeline that balances high-throughput video processing with the computational demands of real-time person re-identification.

3.2.6 Configuration summary

The following tables summarize the key configuration parameters for different components of the Kafka cluster deployment in the Re-ID system.

a, Kafka Cluster and Topic Configuration

Table 3.1: Kafka Cluster and Topic Configuration

Parameter	**Value**
Topic Name	`reid_input`
Number of Partitions	6
Replication Factor	3
Min In-Sync Replicas	2
Cleanup Policy	Delete
Retention Time	1 day (86400000 ms)
Max Size on Disk	20 GB
Maximum Message Size	20 MB
Broker 1	kafka1:29092 (Internal), :9092 (External)
Broker 2	kafka2:29093 (Internal), :9093 (External)
Broker 3	kafka3:29094 (Internal), :9094 (External)
Inter-Broker Protocol	PLAINTEXT
Zookeeper Connection	zookeeper:2181
Zookeeper Configuration	Single node

This cluster configuration at Table 3.1 provides robust fault tolerance for continuous video streaming. The 6 partitions enable parallel processing of up to 6 camera feeds simultaneously, while the 3-broker setup with replication factor 3 ensures zero downtime—even if one broker fails, the system continues operating seamlessly. The 20 MB message limit accommodates high-resolution video frames, and the 1-day retention allows for replay and debugging of Re-ID results. The 20 GB disk limit prevents storage overflow while maintaining sufficient buffer for peak traffic periods.

b, Producer Configuration (Edge Devices)

Table 3.2: Kafka Producer Configuration for Edge Devices

Parameter	Value
Key Serializer	UTF-8 encoding for device identifiers
Value Serializer	Custom Avro-based serialization
Acknowledgment	`acks=1` (leader acknowledgment)
Batch Size	16384 bytes (16 KB)
Linger Time	10 ms (batch collection delay)
Max In-Flight Requests	5 per connection
Partition Strategy	Round-robin distribution

This producer configuration at Table 3.2 optimizes edge device performance for real-time video streaming. The `acks=1` setting provides fast acknowledgments (reducing latency) while ensuring reliable delivery-critical for maintaining smooth 12-20 FPS video streams. The 10 ms linger time allows efficient batching without noticeable delay, improving network utilization by up to 3$\times$ compared to individual message sending. Setting `max_in_flight_requests_per_connection = 5` enables pipeline processing, allowing edge devices to continue sending frames while waiting for acknowledgments, preventing video stuttering during network fluctuations.

c, Consumer Configuration (Central Server)

Table 3.3: Kafka Consumer Configuration for Central Server

Parameter	Value
Auto Offset Reset	`earliest` (start from beginning)
Auto Commit	`True` (automatic offset management)
Group ID	Configurable consumer group identifier
Session Timeout	30000 ms (30 seconds)
Heartbeat Interval	3000 ms (3 seconds)
Max Poll Interval	300000 ms (5 minutes)
Retry Backoff	100 ms (minimal retry delay)
Reconnect Backoff	1000 ms (1 second)
Fetch Min Bytes	1024 KB (1 MB minimum fetch)
Fetch Max Bytes	102400 KB (100 MB maximum fetch)
Max Partition Fetch	102400 KB (100 MB per partition)

This consumer configuration in table 3.3 is specifically tuned for GPU-intensive Re-ID processing. The 5-minute max poll interval prevents consumer timeouts during heavy computation (feature extraction, similarity matching), while the large

fetch sizes (100 MB) reduce network overhead for high-resolution video frames. The `earliest` offset reset ensures no frames are lost during server restarts—crucial for maintaining tracking continuity. The extended session timeout (30 seconds) accommodates variable GPU processing times without triggering unnecessary rebalancing, maintaining stable partition assignments for consistent camera-to-consumer mapping.

3.3 Edge devices

The edge device processing pipeline presented in Figure 3.7 illustrates a compact and efficient architecture designed specifically for CPU-based edge devices equipped with a single-core CPU operating at 3.5 GHz and limited memory capacity of 512 MB RAM. This streamlined approach optimally addresses the constraints of computational resources while performing real-time video analytics directly at the network edge.

The pipeline operates through several integrated components that work together to achieve efficient video processing:

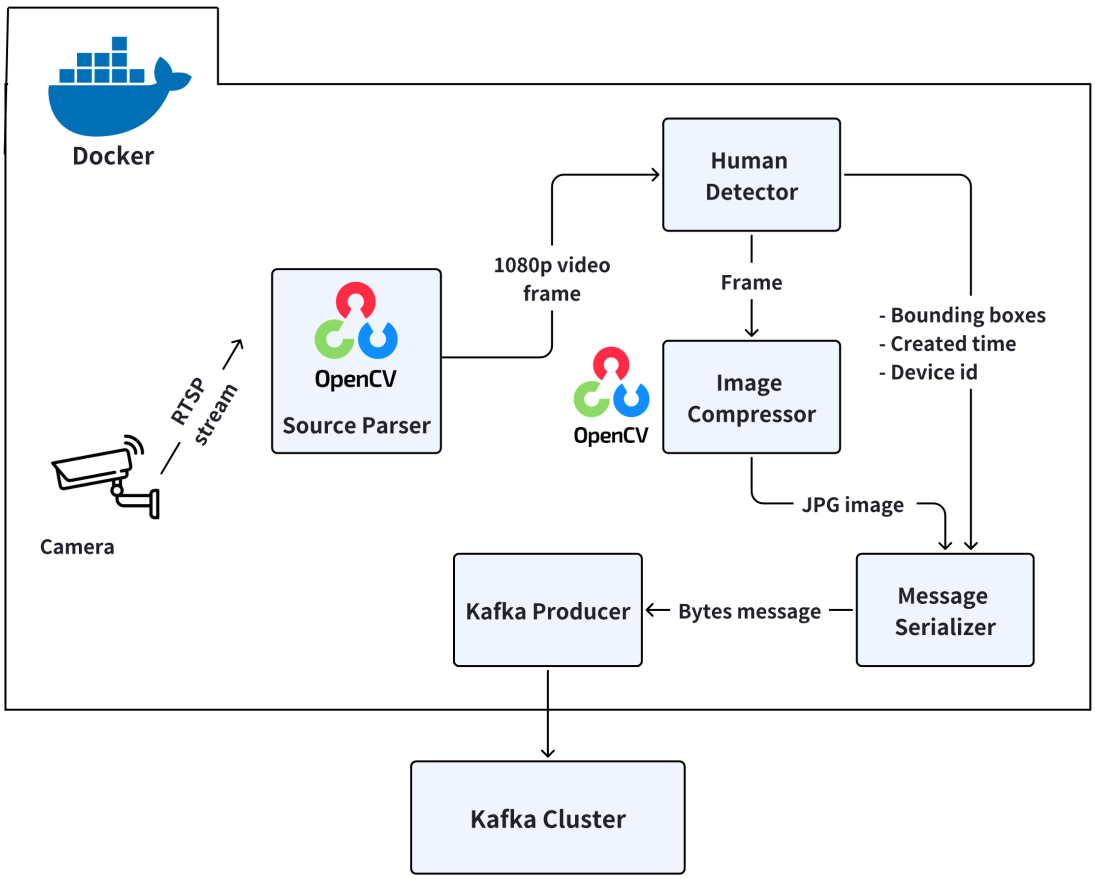

Figure 3.7: Edge devices processing pipeline

- *Source parser*: This component handles input from RTSP cameras delivering

live video streams or can be loaded from video sources. Both inputs are uniformly processed through an OpenCV-based source parser that standardizes the video streams into 1080p frames, thus simplifying subsequent processing steps and minimizing resource overhead.

- *Human detector*: The standardized video frames are then processed by a lightweight yet robust YOLOv11 object detection model, excellently suited for human detection. This human detector module identifies human subjects within each frame, generating precise bounding box coordinates and essential metadata such as timestamp and device identifier.
- *Image Compressor*: To further optimize resource usage, the original video frames undergo simultaneous compression using OpenCV's image processing capabilities, producing compressed JPG images. This significantly reduces the data size but still feasible for constrained CPU environments, without compromising analytical quality.
- *Message Serializer*: Detection results along with the compressed images are efficiently structured into Avro schema serialized messages, significantly improving serialization speed and resource efficiency.
- *Kafka producer*: Finally, these structured messages are published to Kafka `reid_input` topic via Kafka producer framework, enabling asynchronous, real-time communication with centralized server without being blocked by the centralized server processing.

This targeted approach ensures minimal resource consumption, reduced network bandwidth requirements, prompt local analytics responses, and increased scalability by distributing the video processing workload across multiple constrained CPU-based edge devices.

3.3.1 Human detector

As illustrated in Figure 3.7, human detector is the first component in the edge device processing pipeline, after receiving the output from source parser. It is responsible for detecting humans in the video frames and generating bounding box coordinates and essential metadata such as timestamp and device identifier.

The YOLOv11n model deployed within the edge device processing pipeline serves as a robust and resource-efficient solution for real-time human detection, optimized specifically for environments with constrained computational resources.

a, Inference configuration

The inference setup for YOLOv11n is carefully optimized to balance accuracy and computational efficiency, particularly suitable for CPU-based edge devices. Key configurations include:

- *Confidence Threshold:* **0.25**
  This threshold is specifically chosen to complement ByteTrack, a multi-object tracking algorithm used in the subsequent pipeline stage. ByteTrack effectively manages object disappearance by utilizing two stages of tracklet association: one for high-confidence tracklets and another for low-confidence tracklets. The confidence threshold of 0.25 provides a balanced trade-off, ensuring objects are consistently tracked without introducing excessive noise from low-confidence detections.
- *Class ID:* **0**
  Although YOLO is trained on large-scale datasets like COCO, containing 80 classes, our pipeline exclusively focuses on detecting humans. Hence, we restrict detection to class ID 0, which corresponds specifically to humans, improving efficiency and simplifying subsequent processing.
- *Inference Device:* **CPU**
  The inference is conducted entirely on the CPU, aligning with the constrained computing resources typically available in edge deployment scenarios.s

Post-inference, only the following essential detection data are extracted for efficient downstream processing:

```
{
    "bbox": box.xyxy[0].tolist(),
    "confidence": float(box.conf),
    "class_id": int(box.cls),
}
```

Figure 3.8: Extracted bounding box information after inference

The bounding box information follows a structured format specifically designed for optimal performance:

- **bbox**: Coordinates are extracted in xyxy format, representing the top-left corner (x_1, y_1) and bottom-right corner (x_2, y_2) of the detected human bounding box. This format is particularly efficient for subsequent tracking algorithms and

provides direct compatibility with ByteTrack's input requirements.

- **confidence**: The detection confidence score is converted to a floating-point value ranging from 0.0 to 1.0, indicating the model's certainty about the presence of a human in the detected region. This score is crucial for the two-stage association process in ByteTrack.
- **class_id**: Since the system focuses exclusively on human detection, this field consistently contains the value 0 (corresponding to the "person" class in COCO dataset). While redundant in single-class scenarios, maintaining this field ensures compatibility with multi-class detection pipelines and facilitates future system extensions.

This minimal yet comprehensive data structure ensures that only essential information is transmitted to the Kafka cluster, significantly reducing message payload size while preserving all necessary data for accurate tracking and Re-ID processing at the centralized server.

b, Introducing to ONNX model format

For further optimization, YOLOv11 is converted from the PyTorch (`.pt`) format to ONNX (Open Neural Network Exchange), enabling interoperability and improved inference performance in CPU-based deployment environments. The ONNX format facilitates dynamic and simplified computation graphs, significantly enhancing processing efficiency and reducing latency during real-time detection tasks on the edge device.

The conversion process is automated, utilizing YOLO's built-in export function that dynamically simplifies the computation graph to maintain a lightweight and efficient model suitable for low-resource environments:

```
model.export(
    format="onnx",
    dynamic=True,
    simplify=True,
)
```

Figure 3.9: ONNX model conversion command

Here, the `dynamic=True` flag ensures that the model can handle varying input sizes, while the `simplify=True` flag simplifies the model to reduce unnecessary operations, resulting in a more efficient and streamlined model. The output ONNX model is saved in the `yolo11n.onnx` file.

c, Introducing to OpenVINO model format

Alternatively, for Intel hardware-based edge deployments, YOLOv11 is converted into the OpenVINO model format. This conversion leverages Intel's OpenVINO toolkit, which provides further optimizations tailored specifically for CPU inference on Intel architectures. OpenVINO models offer reduced inference time and enhanced resource utilization, ensuring optimal performance within the specified hardware constraints of single-core CPUs and limited RAM.

The OpenVINO conversion is similarly automated using YOLO's export utilities, producing optimized models capable of handling real-time analytics tasks efficiently under the given hardware limitations:

```
model.export(
    format="openvino",
    dynamic=True,
    simplify=True,
)
```

Figure 3.10: OpenVINO model conversion command

Similar to the ONNX conversion, the `dynamic=True` flag ensures that the model can handle varying input sizes, while the `simplify=True` flag simplifies the model to reduce unnecessary operations, resulting in a more efficient and streamlined model. The output OpenVINO model is saved inside a folder which contains 3 files:

- `yolo11n.xml`
- `yolo11n.bin`
- `metadata.yaml`

To determine the optimal model format for edge deployment scenarios, a comprehensive comparative evaluation is conducted across multiple performance metrics in Section 4.3.1. This evaluation systematically analyzes inference latency, memory consumption, CPU utilization, and detection accuracy across PyTorch (.pt), ONNX, and OpenVINO formats under the specified hardware constraints. The results provide empirical evidence to guide format selection based on specific deployment requirements and hardware configurations.

3.3.2 Image compressor with OpenCV

As mentioned in Section b, the original video frames of FullHD resolution (1920x1080) have a size of around 2MB. If we keep the original image quality, the

total message size would be too large for practical network bandwidth and storage. Therefore, there is a need to compress the image quality.

OpenCV provides a simple and effective way to compress the image quality by using the `cv2.imencode` function. The function takes a quality parameter which ranges from 0 to 100, where 0 is the highest compression and 100 is the original image quality.

```
cv2.imencode(".jpg", frame, [cv2.IMWRITE_JPEG_QUALITY, 70])
```

Image compression with OpenCV

The function `cv2.imencode` requires 3 main parameters:

- `".jpg"`: The image format. Here, we use the JPEG format for efficient compression.
- `frame`: The image frame to be compressed in numpy array format.
- `IMWRITE_JPEG_QUALITY`: The quality parameter. Here, we set the quality to 70, which is a good balance between compression and image quality.

(a) Quality 10 - 62KB

(b) Quality 40 - 116KB

(c) Quality 70 - 168KB

(d) Quality 100 - 747KB

Figure 3.11: Comparison of JPEG compression quality levels showing the trade-off between file size and image quality. Quality 10 provides maximum compression - only 62KB, but with noticeable quality loss, while quality 100 maintains original image quality with larger file size - 747KB. Quality of 70 provides a good balance between compression and image quality - 168KB.

3.3.3 Messages serialization

As mentioned in Section b, the detection results along with the compressed images are structured into Avro schema serialized messages before being published

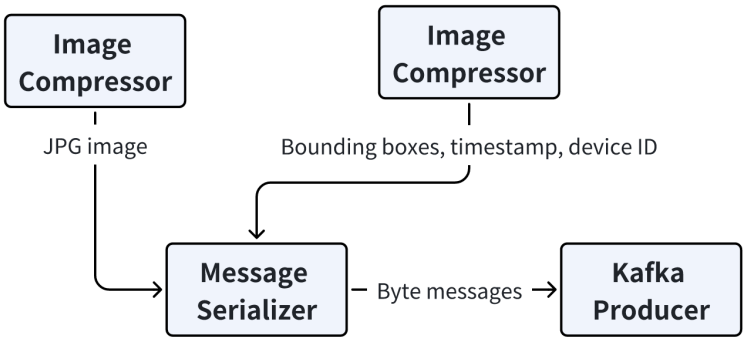

Figure 3.12: Message serialization flow

to Kafka.

```
def serialize_message(self, detection_result, image):
    writer = DatumWriter(self.avro_schema)
    bytes_writer = python_io.BytesIO()
    encoder = io.BinaryEncoder(bytes_writer)
    writer.write(message, encoder)
    return bytes_writer.getvalue()
```

Figure 3.13: Message serialization with Avro schema, using `DatumWriter` and `BinaryEncoder`

The `DatumWriter` class take the parsed the Avro schema and create a writer object. The `bytes_writer` is a buffer to store the serialized message. The `encoder` is a binary encoder to encode the message. The `writer.write` method takes the message and the encoder, and write the message to the buffer. Finally, the `bytes_writer.getvalue()` method returns the serialized message in bytes, before being published to Kafka cluster via **Kafka producer** framework.

3.3.4 Containerization with Docker

The edge device application is containerized using Docker to ensure consistent deployment across different hardware platforms and operating systems. The containerization strategy employs a multi-stage build approach that optimizes both build time and final image size, which are critical considerations for resource-constrained edge environments.

a, Multi-stage build architecture

The Docker configuration utilizes a two-stage build process designed to maximize efficiency and minimize the final container footprint. This approach addresses the specific constraints of edge computing environments where storage space and deployment speed are critical considerations.

```
# Build stage
FROM ghcr.io/astral-sh/uv:python3.11-bookworm-slim AS builder
ENV UV_COMPILE_BYTECODE=1 \
    UV_LINK_MODE=copy \
    UV_PYTHON_DOWNLOADS=0
WORKDIR /app
# Copy dependency files first for better caching
COPY pyproject.toml uv.lock ./
# Install dependencies in virtual environment
RUN uv sync --locked --no-dev
# Production stage
FROM python:3.11-slim-bookworm AS production
# Install system dependencies required for OpenCV
RUN apt-get update && apt-get install -y \
    libgl1-mesa-glx \
    libglib2.0-0 \
    libsm6 \
    libavcodec-dev \
    libavformat-dev \
    libswscale-dev \
    libjpeg-dev \
    && rm -rf /var/lib/apt/lists/*
WORKDIR /app
# Copy virtual environment from builder
COPY --from=builder /app/.venv /app/.venv
# Copy source code
COPY . .
# Set environment variables
ENV PATH="/app/.venv/bin:$PATH" \
    PYTHONPATH=/app \
    PYTHONUNBUFFERED=1
# Set the entrypoint to run main.py with arguments
ENTRYPOINT ["python", "-m", "src"]
```

Multi-stage Docker build configuration for edge deployment

b, Build stage optimization

The first stage, designated as the **builder** stage, leverages the specialized `uv` base image from Astral, which provides an ultra-fast Python package installer and resolver. This stage is specifically optimized for dependency management:

- **Efficient dependency resolution**: The `uv` tool significantly reduces dependency installation time compared to traditional `pip`, which is particularly beneficial during development iterations when dependencies may change frequently.

- **Layer caching optimization**: By copying only `pyproject.toml` and `uv.lock` files first, Docker can effectively cache the dependency installation layer. This means that subsequent builds will only reinstall dependencies if these files change, dramatically reducing build times during development.
- **Bytecode compilation**: The `UV_COMPILE_BYTECODE=1` environment variable ensures that Python bytecode is compiled during the build process, improving runtime performance on edge devices where CPU resources are limited.
- **Production-only dependencies**: The `--no-dev` flag excludes development dependencies from the final build, reducing the virtual environment size and eliminating unnecessary packages that could introduce security vulnerabilities or consume valuable storage space.

c, Production stage configuration

The second stage creates the final production image with several key optimizations for edge deployment:

- **Minimal base image**: The `python:3.11-slim-bookworm` base image provides the essential Python runtime while maintaining a small footprint, crucial for edge devices with limited storage capacity.
- **System dependencies**: Essential libraries for OpenCV and video processing are installed, including:
  - `libgl1-mesa-glx`: OpenGL support for computer vision operations
  - `libavcodec-dev`, `libavformat-dev`, `libswscale-dev`: FFmpeg libraries for video codec support
  - `libjpeg-dev`: JPEG compression/decompression capabilities
- **Virtual environment transfer**: The complete virtual environment is copied from the builder stage, ensuring all dependencies are properly isolated and configured without requiring reinstallation.
- **Runtime optimization**: Environment variables are configured to optimize Python execution:
  - `PYTHONUNBUFFERED=1`: Ensures immediate output flushing for real-time logging
  - `PYTHONPATH=/app`: Facilitates proper module resolution

d, Caching benefits and deployment efficiency

This multi-stage approach provides several significant advantages for edge deployment scenarios:

- **Reduced image size**: By excluding build tools and development dependencies from the final image, the production container is significantly smaller, reducing deployment time and storage requirements on edge devices.
- **Improved build caching**: Docker's layer caching mechanism ensures that dependency installation only occurs when `pyproject.toml` or `uv.lock` changes, dramatically reducing build times during iterative development from minutes to seconds.
- **Reproducible builds**: The locked dependency file (`uv.lock`) ensures identical dependency versions across all deployments, eliminating version conflicts and ensuring consistent behavior across different edge devices.
- **Security optimization**: The minimal production image reduces the attack surface by excluding unnecessary development tools and libraries, enhancing security for edge deployments in potentially untrusted environments.
- **Resource efficiency**: The optimized container consumes less memory and storage, allowing more efficient utilization of the limited resources available on edge devices while maintaining full functionality.

This containerization strategy ensures that the edge application can be consistently deployed across diverse hardware platforms while maintaining optimal performance within the specified constraints of single-core CPUs and 512 MB RAM limitations.

3.4 Centralized server

The centralized server architecture represents a most principal component in the hybrid edge-server person Re-ID system, serving as the computational backbone for resource-intensive operations that exceed the capabilities of edge devices. While the CPU-based edge devices handle distributed object detection and initial preprocessing, the centralized server leverages its GPU computing resources to perform sophisticated model inference tasks, including feature extraction and gender classification. Additionally, the server manages vector database operations for identity storage and retrieval, coordinates multi-consumer message processing from Kafka streams, and maintains the overall system state for cross-camera identity matching, as illustrated in Figure 3.14. This centralized microservice-based approach enables the system to achieve optimal resource utilization by offloading computationally demanding tasks from resource-constrained edge devices to a dedicated server environment equipped with

specialized hardware acceleration. The server's architecture is designed to handle concurrent processing of multiple video streams while maintaining real-time performance requirements essential for practical surveillance applications.

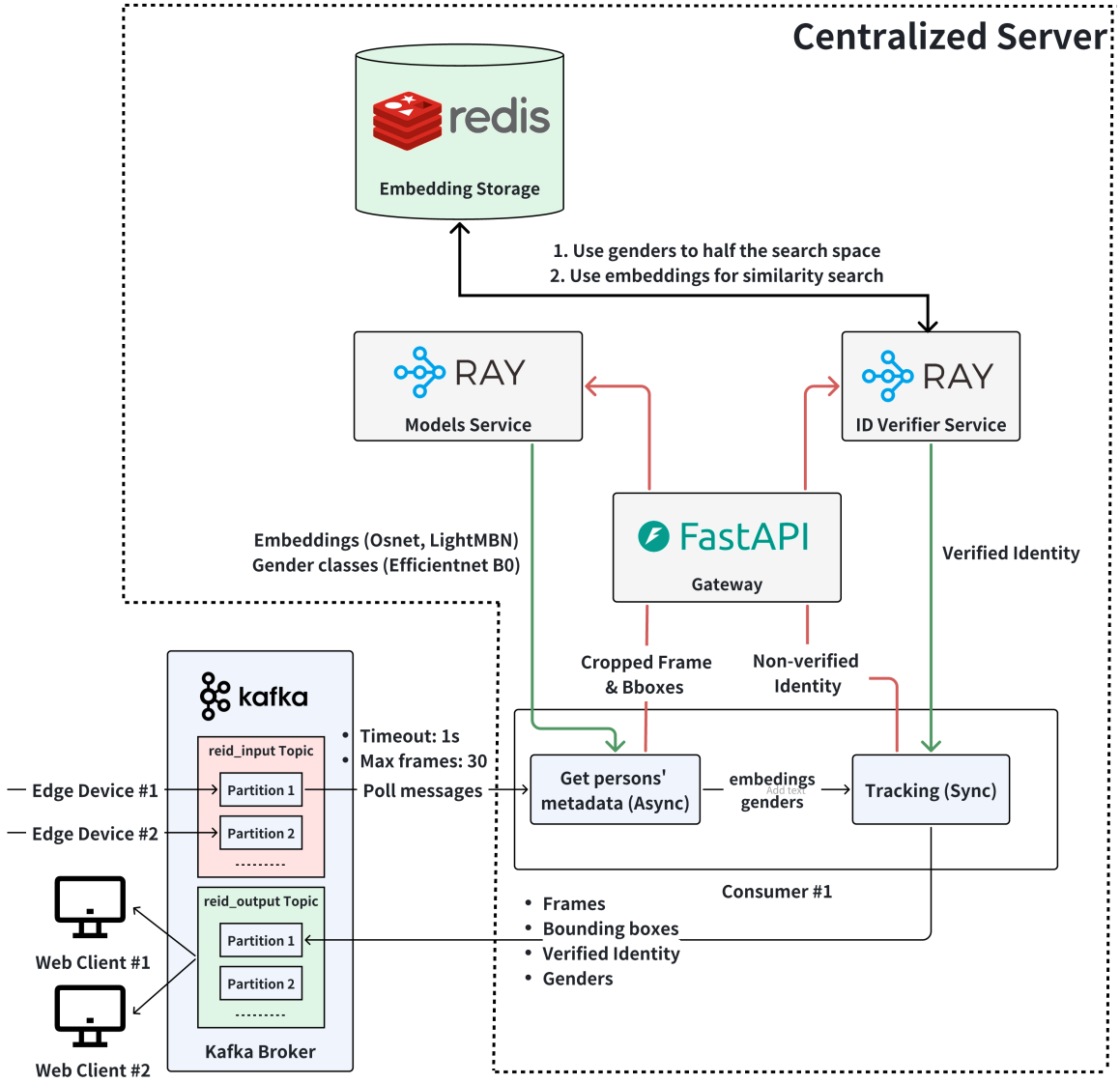

Figure 3.14: Centralized server architecture. The server is equipped with a decent GPU and CPU, and it is responsible for performing sophisticated model inference tasks, vector database operations, and multi-consumer message processing.

The comprehensive system data flow and processing pipeline architecture is detailed in Section 3.4.1, which provides an in-depth analysis of the interconnected components and their operational mechanisms within the centralized server framework.

3.4.1 System data flow and processing pipeline

The centralized server architecture orchestrates a complex data flow that transforms raw video streams into actionable identity verification results through multiple interconnected services. As depicted in Figure 3.14, the processing pipeline begins when edge devices capture and preprocess video frames, transmitting video frames,

cropped person detections, and bounding box coordinates through Kafka message queues to the centralized server. Meanwhile, multiple consumers are polling messages from the topic `reid_input` to starting the processing pipeline, which will be discussed more in detail in the next section.

a, Messages polling from Kafka

Each partition of the Kafka topics corresponds to a specific camera feed, and each consumer group is responsible for processing messages from a single partition, ensuring that message ordering is preserved, which is a critical requirement for the tracking module of the Re-ID system. As illustrated in Figure 3.14, Consumer #1 represents the data processing pipeline that serves as a template for additional consumers. Each consumer operates within an event loop that fetches batched messages (maximum 30 messages) from its assigned Kafka topic partition every 1 second. This batched processing approach significantly reduces computational overhead compared to continuous message fetching, while maintaining near real-time processing capabilities essential for surveillance applications.

b, Get person's metadata (Asynchronous)

After receiving the batch of messages, the system processes each message (frame) to get the person's metadata through a structured pipeline:

1. Starting the event loop to asynchrously process the batch of messages (frames) from the Kafka topic `reid_input`.
2. For each frame containing cropped person detections and bounding box coordinates, the system extracts person regions and computes two essential properties (referred to as person metadata) asynchronously:
   - *Embedding vector*: A numerical feature representation of the person region used for similarity searches in the vector database to determine whether the person already exists in the system. In the proposed system, OSNet and LightMBN models are used to extract visual embeddings from person regions, which are then stored in the vector database for later retrieval and matching.
   - *Gender classification*: The predicted gender of the person (male or female), enabling gender-based search space partitioning that reduces computational complexity by approximately 50% by querying only relevant gender subsets during similarity matching operations.
3. When all frames in the batch are processed, the results are sorted by their original timestamps to maintain temporal order before proceeding to the tracking

phase.

During the asynchronous processing, the system uses the **Models Service** via the **FastAPI Gateway** - acting as the central coordination layer that orchestrates the processing pipeline between Kafka consumers and Ray Serve services - to extract embeddings and classify gender.

After receiving the results from the **Models Service**, which is the embeddings and gender classes (male or female) for all the person regions of each frame, the "Get person's metadata (Async)" module will prepare them as the metadata for each frame before forwarding them to the next module: "Tracking (Sync)", which will be discussed in detail in the next section.

c, Tracking (Sync)

The sorted embeddings and gender classifications are then forwarded to the customized **Tracking (Sync)** module, which iterates through each frame in correct timestamp order. The tracking module - ByteTrack uses detection results and embeddings to perform association with previous tracklets, returning a list of person IDs for each frame. Crucially, only newly discovered IDs that do not exist in previous tracklets are considered "non-verified" and require database verification. This is because tracking algorithms can only maintain ID persistence in the short term and cannot determine whether a person actually existed in the system previously.

For these non-verified IDs, the tracking module requests the **ID Verifier Service** to query the Redis vector database using gender-based search space partitioning and embedding similarity search. Once verified identities are returned, the tracking module updates its tracklet data with the correct IDs, ensuring that subsequent frames do not require repeated database queries for the same person, as the tracking module only consults the ID verifier service when encountering new, unrecognized identities.

After the tracking module has processed all frames in the batch, the results are ready to be published back to the Kafka output topics.

d, Publishing results to output topic

Processed results, including verified identities, gender classifications, bounding box coordinates, and confidence scores, are published back to Kafka output topics using the same partitioned structure. This bidirectional communication pattern enables multiple web clients to subscribe to specific camera feeds or system-wide notifications, supporting diverse use cases from real-time monitoring dashboards to alert systems and analytical reporting applications.

The architecture's microservice design ensures fault tolerance and independent scaling, where individual services can be replicated or upgraded without affecting the overall system operation, making it suitable for production deployment in enterprise surveillance environments.

For a comprehensive understanding of the individual services comprising the centralized server architecture, the subsequent sections provide detailed technical analysis, commencing with the Models Service implementation.

3.4.2 Models service

This section examines the implementation details of the **Models Service**, which performs feature extraction and gender classification on detected person regions. The service is built using Ray Serve, a scalable model serving library designed for building high-performance online inference APIs. Ray Serve provides several key features and performance optimizations specifically beneficial for serving PyTorch models, including dynamic request batching, multi-node and multi-GPU serving capabilities, and automatic load balancing across distributed computing resources.

a, Ray Serve configuration

The Ray Serve configuration defines the deployment parameters and resource allocation for the Models Service. This setup establishes a distributed serving environment optimized for concurrent model inference across multiple GPU-accelerated workers, as demonstrated in the following `config.yaml` file, depicted in Figure 3.15.

Network Configuration: The service is configured to accept HTTP requests on port 8000 and gRPC requests on port 9000, with the proxy deployed on every node (`EveryNode`) to ensure load distribution and minimize network latency. The `0.0.0.0` host binding enables external access from edge devices and other system components.

Resource Allocation Strategy: The deployment specifies 4 replicas of the ModelService, each allocated 2.0 CPU cores and 0.25 GPU units. This fractional GPU allocation enables efficient GPU memory sharing, allowing multiple model inference processes to utilize the same GPU simultaneously. With 4 replicas sharing GPU resources, the system can achieve optimal GPU utilization while maintaining parallel processing capabilities.

```yaml
proxy_location: EveryNode
http_options:
    host: 0.0.0.0
    port: 8000
grpc_options:
    port: 9000
    grpc_servicer_functions: []
logging_config:
    encoding: TEXT
    log_level: INFO
    logs_dir: null
    enable_access_log: true
    additional_log_standard_attrs: []
applications:
    - name: thesis-model-serving
    route_prefix: /
    import_path: src.main.model_service
    runtime_env: {}
    deployments:
    - name: ModelService
        num_replicas: 4
        max_ongoing_requests: 32
        ray_actor_options:
        num_cpus: 2.0
        num_gpus: 0.25
```

Figure 3.15: Ray Serve configuration file

The multiple replica strategy addresses a fundamental challenge in serving lightweight neural networks on GPU hardware. The deployed models are relatively compact: OSNet contains approximately 2 million parameters (8MB model size), while EfficientNet B0 comprises around 5.3 million parameters (21MB model size). These small model footprints significantly underutilize GPU computational resources when processed sequentially through traditional approaches such as iterative for-loops over bounding box detections.

Sequential processing of individual person regions results in GPU underutilization due to insufficient computational load per inference operation. Modern GPUs are designed for high-throughput parallel computation, but lightweight models cannot maximize usage of the available CUDA cores and tensor processing units. This mismatch leads to a processing bottleneck where the system becomes constrained by sequential execution rather than GPU computational capacity, resulting in suboptimal requests per second (RPS) performance that plateaus well below the hardware's theoretical maximum.

By deploying multiple replicas with fractional GPU allocation, the system transforms the inference pattern from sequential to parallel execution. Each replica can simultaneously process different batches of person detections, effectively multiplying the concurrent inference operations. This approach maximizes GPU utilization by ensuring that multiple inference operations are executed in parallel across different CUDA streams, saturating the GPU's computational resources and achieving significantly higher RPS throughput than single-replica deployments.

The 4-replica configuration with 0.25 GPU allocation per replica ensures that the total GPU memory and computational resources are fully utilized while preventing resource contention. This design enables the system to handle concurrent inference requests from multiple camera feeds efficiently, transforming the GPU from an underutilized resource in sequential processing to a fully optimized parallel processing engine capable of meeting real-time surveillance demands.

Concurrency Management: Each replica is configured to handle a maximum of 32 ongoing requests (`max_ongoing_requests: 32`), providing a total system capacity of 128 concurrent inference requests across all replicas. This configuration balances memory consumption with throughput requirements, preventing GPU memory overflow while maximizing processing efficiency.

Request Batching Benefits: The combination of multiple replicas and concurrent request handling enables Ray Serve's dynamic batching capabilities to automatically group incoming requests for batch inference. This batching mechanism significantly improves GPU utilization by processing multiple person regions simultaneously, reducing per-request inference time and increasing overall system throughput.

Fault Tolerance and Scaling: The multi-replica deployment provides inherent fault tolerance, where individual replica failures do not compromise system availability. Additionally, the configuration supports horizontal scaling by adjusting the `num_replicas` parameter based on workload demands, enabling dynamic resource allocation in response to varying inference loads from multiple camera feeds.

This configuration strikes an optimal balance between resource utilization, processing latency, and system reliability, ensuring that the **Models Service** can efficiently handle the computational demands of real-time person Re-ID applications across distributed edge-server architectures.

b, Serving feature extraction module with Ray Serve

The feature extraction component represents the core computational module of the Models Service, responsible for transforming raw person detection images

into numerical feature representations suitable for similarity matching and identity verification. The implementation leverages two state-of-the-art person Re-ID models-OSNet and LightMBN-each optimized for different aspects of feature extraction performance and accuracy.

1. **Cold start problem**:
   The cold start problem in deep learning model serving refers to the initial latency penalty incurred during the first inference request, caused by GPU memory allocation, CUDA kernel compilation, and model weight loading operations. To mitigate this issue, the Models Service implements a comprehensive warmup strategy during initialization through the `_warmup()` method. This approach creates dummy input tensors with the exact dimensions required by each model: `torch.randn(1, 3, 256, 128)` for OSNet and LightMBN models, and `torch.randn(1, 3, 224, 224)` for the EfficientNet gender classification model. By executing forward passes with these dummy inputs during service startup, the system pre-allocates GPU memory, compiles necessary CUDA kernels, and initializes all computational graphs, ensuring that subsequent real inference requests experience minimal latency without the overhead of first-time GPU operations.

2. **Image preprocessing**:
   Initially, image bytes from the API are in raw bytes format, so we need to decode them to `Image.PIL` format using
   `BytesIO` and the function `Image.open()`. The preprocessing pipeline ensures consistent input format by converting non-RGB images to RGB mode using `Image.convert("RGB")`, which is essential for maintaining color channel consistency across different input image formats.

   The system implements two distinct preprocessing pipelines optimized for different model requirements:

   Embedding models (OSNet and LightMBN): Images are resized to 256×128 pixels to match the input dimensions expected by person Re-ID models, which are specifically designed for person-centric aspect ratios. The preprocessing applies ImageNet normalization with mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225] to ensure compatibility with pre-trained model weights.

   Classification model (EfficientNet): Images are resized to 224×224 pixels, the standard input size for EfficientNet architectures, and normalized using the same ImageNet statistics to maintain consistency with pre-training data distribution.

Both pipelines convert images to PyTorch tensors using `transforms.ToTensor()`, which automatically scales pixel values from [0, 255] to [0, 1] range and reorders dimensions from HWC (Height-Width-Channel) to CHW (Channel-Height-Width) format required by PyTorch models.

3. **OSNet model**:
   The OSNet (Omni-Scale Network) model serves as the primary feature extraction backbone, initialized with `osnet_x1_0` architecture containing 1000 output classes for comprehensive feature representation. The model is loaded with pre-trained weights (`pretrained=True`) and configured with softmax loss for optimal person Re-ID performance. OSNet's distinctive architecture employs omni-scale convolutions that capture multi-scale features simultaneously, making it particularly effective for person Re-ID tasks where scale variations are common due to different camera distances and viewing angles. The model processes 256×128 input images and outputs feature vectors that capture discriminative person characteristics essential for cross-camera identity matching.

4. **LightMBN model**:
   The LightMBN (Lightweight Multi-Branch Network) model provides an alternative feature extraction approach optimized for computational efficiency while maintaining competitive accuracy. Initialized with 512 feature dimensions and configured without activation mapping (`activation_map=False`) for streamlined inference, LightMBN outputs features in shape [batch_size, 512, 7], where the 7 components represent different spatial regions of the person image. The system applies average pooling across these 7 components using `features.mean(dim=2)` to produce a consolidated 512-dimensional feature vector, effectively combining multi-region information into a single representative embedding suitable for similarity calculations and database storage.

5. **API gateway implementation**:
   The Models Service exposes three distinct API endpoints, each optimized for different usage patterns and performance requirements:

   - **Single image API** (`/embedding`): Designed for low-latency applications requiring immediate response, this endpoint processes individual images through the `extract_features_single` method. It prioritizes response time over throughput, making it suitable for interactive applications or real-time processing scenarios where each request must be handled independently without waiting for batch formation.
   - **Dynamic batching API** (`/embedding/batch`): Leverages Ray Serve's

automatic batching capabilities with `@serve.batch` decorator configured for maximum batch size of 32 images and 10ms timeout (`batch_wait_timeout_s=0.01`). This endpoint automatically groups concurrent requests into batches, optimizing GPU utilization by processing multiple images simultaneously while maintaining acceptable latency. The dynamic batching approach balances throughput optimization with responsiveness, automatically adapting to varying request loads.

- **True batching API** (`/embedding/true-batch`): Processes multiple images submitted as a single request through the `List[UploadFile]` parameter, enabling clients to explicitly control batch composition. For PyTorch models, the system utilizes matrix calculations to stack individual image tensors into a single batch tensor using `torch.cat(processed_images, dim=0)`, then performs inference on the entire batch vector simultaneously. This approach maximizes computational efficiency by leveraging GPU's parallel processing capabilities for matrix operations, significantly reducing inference time compared to sequential processing. The batched inference returns results for all images in a single forward pass, eliminating the overhead of individual request processing and maximizing GPU computational throughput for scenarios where multiple images are available simultaneously, such as processing video frame sequences or bulk image analysis tasks.

Each API endpoint returns structured responses containing feature vectors, dimensional information, and processing metadata, enabling clients to select the most appropriate processing mode based on their specific latency and throughput requirements.

c, Serving gender classification module with Ray Serve

The gender classification component serves as an optimization module within the Models Service, designed to reduce computational complexity in identity verification by enabling gender-based search space partitioning. This approach leverages the biological distinction between male and female individuals to partition the vector database, effectively reducing the search space by approximately 50% during similarity matching operations when implemented. The implementation utilizes a custom-trained EfficientNet B0 model specifically optimized for person gender classification tasks.

Model Architecture and Configuration: The gender classification model is built upon the EfficientNet B0 architecture, a compound scaling method that uniformly

scales network width, depth, and resolution with a set of fixed scaling coefficients. The model is initialized through the `GenderClassificationModel` class and configured to process 224×224 pixel RGB images, matching the standard input dimensions for EfficientNet architectures. The model outputs binary classification logits representing male and female categories, with softmax activation applied to generate probability distributions for confidence scoring.

Custom Training Approach: Unlike the pre-trained person Re-ID models (OSNet and LightMBN), the gender classification model represents a custom-trained solution developed specifically for this surveillance system.

Preprocessing Pipeline: The gender classification preprocessing follows the same ImageNet normalization standards as the embedding models but with distinct dimensional requirements. Input images are resized to 224×224 pixels to match EfficientNet's expected input dimensions, converted to RGB format for color consistency, and normalized using mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225]. This preprocessing ensures optimal compatibility with the model's learned feature representations while maintaining consistency with standard computer vision practices.

API Implementation: The gender classification functionality is exposed through the `/gender/classify` endpoint, which processes individual images and returns structured predictions including the predicted gender label, confidence score, and probability distributions for both male and female categories. The API implementation follows the same preprocessing pipeline as other model endpoints, utilizing the `preprocess_single` method with "efficientnet" model specification to ensure appropriate image transformations.

The inference process applies softmax activation to the model's logits using `F.softmax(logits, dim=1)` to generate normalized probability distributions, followed by `torch.argmax` to determine the predicted class. The system maps numerical predictions to human-readable labels using a predefined mapping where class 0 corresponds to "male" and class 1 corresponds to "female". Additionally, the API returns confidence scores extracted from the maximum probability value, enabling downstream systems to implement confidence-based filtering or uncertainty quantification.

Search Space Optimization: The primary motivation for gender classification lies in its ability to significantly reduce computational overhead during identity verification queries. By classifying detected persons into gender categories, the system can partition the vector database and limit similarity searches to gender-specific subsets. This optimization reduces the average number of embedding comparisons

by approximately 50%, directly translating to faster query response times and improved system scalability. The gender-based partitioning strategy is particularly effective in surveillance scenarios where gender distribution is typically balanced, ensuring that both partitions contribute meaningfully to search space reduction without creating significant load imbalances.

3.4.3 ID verifier service

The ID verifier service implements an identity verification pipeline that combines gender-based search space partitioning with embedding similarity search to determine whether newly detected persons correspond to existing identities in the system database. This service operates as an essential component in the person Re-ID pipeline, receiving "non-verified" IDs from the tracking module and returning verified identity assignments with associated confidence scores.

a, Core Functionality and Gender-Based Partitioning

To efficiently process identity verification requests while maintaining high accuracy, the service queries the Redis vector database using a two-stage verification approach:

- First, the service applies gender classification results with confidence scores greater than 0.9 to partition the search space.
- Second, the service performs embedding similarity calculations within the relevant gender subset.

The 0.9 confidence threshold for gender classification is strategically chosen to address several critical challenges in surveillance environments:

Confidence Threshold Rationale: The 0.9 confidence threshold ensures that only high-confidence gender predictions are used for search space partitioning. When confidence falls below this threshold, the system searches across all gender categories, maintaining accuracy at the cost of increased computational overhead.

Fallback Mechanism: For gender confidence below 0.9, the service searches all stored embeddings to prevent missed identity matches. This strategy is crucial for challenging scenarios including posterior view detection (where facial features are absent), severe occlusion cases, extreme lighting conditions, and distance limitations that reduce image resolution. The fallback maintains system reliability while optimizing performance for high-confidence cases.

b, Integration with Tracking Module

As illustrated in Figure 3.14, the ID Verifier Service maintains bidirectional communication with the tracking module through a structured data exchange protocol. The service receives newly detected person embeddings along with their associated

metadata (including gender predictions, bounding box coordinates, and temporal information), processes these through the verification pipeline, and returns verified identity assignments with confidence scores and similarity metrics.

The tracking module subsequently updates tracklet data with verified IDs, ensuring consistent identity maintenance across video frames and camera views. This integration enables the system to maintain identity consistency throughout person trajectories, even when individuals temporarily leave and re-enter camera coverage areas.

c, API Implementation and Performance

The ID Verifier Service exposes a RESTful API through a single endpoint (`/verify`) that processes individual person embeddings and gender predictions, returning comprehensive verification results including identity matches, confidence scores, and similarity metrics. The API is designed for high-throughput processing with the following characteristics:

Request Format: The service accepts JSON payloads containing embedding vectors (512-dimensional for OSNet), gender predictions with confidence scores, and optional metadata such as camera identifiers and timestamps.

Response Structure: The API returns structured responses containing verified identity assignments, similarity scores, matching confidence levels, and additional metadata required for tracking module integration.

3.4.4 Embedding storage

Embedding storage represents a critical component of the identity management system, responsible for persistently storing and retrieving body embeddings of identified individuals. This storage mechanism enables the system to re-identify people across frames and potentially across multiple cameras in distributed surveillance environments. To achieve the required performance and scalability while maintaining cost-effectiveness for deployment on resource-constrained devices, we utilize Redis, a high-performance in-memory data structure store, as the underlying storage engine.

Redis was selected for its exceptional performance characteristics in real-time applications, including sub-millisecond query response times, native support for complex data structures, and efficient memory-based operations that enable rapid embedding comparisons. The system leverages Redis's built-in expiration mechanisms for automatic stale data management and its atomic operations for thread-safe concurrent access in multi-threaded environments. Additionally, Redis provides flexible data persistence options, allowing the system to balance between pure in-memory performance and data durability based on deployment requirements.

a, Redis Configuration and Data Structure

The Redis-based identity management system is configured to optimize performance for real-time identity matching while ensuring efficient memory utilization and data persistence. The configuration encompasses several critical parameters tailored to the specific requirements of the Re-ID system.

Data Structure Design: Each identity is stored as a Redis hash with a structured key format `personid:{identity_id}`, containing comprehensive metadata and embedding information. The data structure includes:

- `identity_id`: Unique identifier for the tracked individual
- `last_embedding`: 512-dimensional normalized vector from OSNet architecture
- `fullbody_bbox`: Bounding box coordinates of the most recent detection
- `body_conf`: Confidence score of the stored detection (0.0-1.0)
- `gender`: Detected gender classification ("male" or "female")
- `gender_confidence`: Confidence score of gender classification (0.0-1.0)
- `last_seen_frame`: Frame number of most recent observation
- `update_count`: Total number of updates for this identity
- `ttl`: Time-to-live value for automatic expiration

Memory Management: Redis operates with configurable memory allocation and automatic expiration policies. Each identity entry is assigned a time-to-live (TTL) of 1000 frames, enabling automatic cleanup of stale identities without manual intervention. The system utilizes Redis's built-in LRU eviction policies to manage memory pressure during peak operational periods.

Persistence Configuration: The system employs Redis's hybrid persistence approach, combining RDB snapshots for periodic data backup with AOF (Append-Only File) logging for transaction durability, ensuring data integrity across system restarts while maintaining optimal performance.

b, Enhanced Retrieval Strategy with Gender-Based Filtering

The retrieval process implements an intelligent multi-stage approach designed to balance accuracy and computational efficiency through metadata-based pre-filtering. The system employs gender classification as a primary filtering mechanism to reduce search space and improve query performance.

Query Construction and Pre-filtering: For each incoming detection requiring identity matching, the system constructs a query embedding using the OSNet architecture,

generating a 512-dimensional normalized feature vector. Before executing similarity comparisons, the system applies gender-based filtering when both the query person and stored identities have high-confidence gender classifications (threshold > 0.9). This pre-filtering mechanism can reduce the search space by approximately 50% in typical scenarios, significantly improving query execution speed and system throughput.

Adaptive Similarity Computation: The system utilizes PyTorch's `F.cosine_similarity` function for consistent similarity calculations, ensuring alignment with the training methodology. The similarity computation process iterates through gender-filtered candidates, calculating cosine similarity scores between the query embedding and stored embeddings. Only matches exceeding the similarity threshold (default: 0.8) are considered valid candidates.

Identity Update Logic: The system implements a sophisticated identity management strategy that distinguishes between verified and unverified tracks:

1. **Verified Identity Updates**: For existing tracked identities, the system updates stored embeddings only when both detection confidence exceeds 0.75 and the new detection demonstrates higher confidence than the currently stored embedding. This selective updating ensures that only high-quality representations are preserved.
2. **Unverified Identity Processing**: For new or unverified tracks, the system performs comprehensive database searches using the gender-filtered approach. When a similarity match above the threshold is found, the system updates the existing identity with the new embedding and remaps the tracker association.
3. **Quality-Based Embedding Replacement**: The system maintains the highest quality embedding for each identity by comparing detection confidence scores. New embeddings replace existing ones only when they demonstrate superior quality metrics, preserving optimal representation fidelity.

Gender Classification Integration: The system integrates gender classification results with a confidence threshold of 0.9 for search space partitioning. High-confidence gender predictions enable efficient filtering, while low-confidence classifications default to comprehensive searches to prevent false exclusions. This adaptive approach ensures robust performance across diverse detection scenarios while maximizing computational efficiency.

c, Performance Optimization

The Redis-based approach provides several performance advantages, including constant-time key lookups, efficient memory-based similarity computations, and automatic memory management through TTL mechanisms. The system's architecture enables horizontal scaling through Redis clustering for high-throughput deployments while maintaining millisecond-level response times for real-time applications.

4.1 Environmental Setup

Our experiments were conducted on a computer featuring an Intel i5-12600K CPU paired with an NVIDIA GeForce RTX 3060 GPU (12 GB of VRAM) to perform inference for the end-to-end ReID system. Each camera node in the system was equipped with a lightweight, cost-effective device configured with a single-core CPU operating at 3.5 GHz and 512MB of RAM.

Table 4.1: Hardware setup on the server and the edge device

Component	Edge Device	Server
CPU	1-core CPU 3.5 GHz	Intel i5 12600K
GPU	None	RTX 3060 12GB
RAM	512MB	32GB

4.2 Dataset

Three distinct datasets were utilized in this thesis to comprehensively evaluate different components of the person re-identification system. The P-DESTRE dataset (Section 4.2.1) served as the primary training source for the gender classification model, providing diverse aerial perspectives with detailed soft biometric annotations. The CUHK03 dataset (Section 4.2.2) was employed for benchmarking the embedding model performance, offering a standardized evaluation framework with challenging outdoor surveillance scenarios. Finally, the BK6I dataset (Section 4.2.3) provided video-based sequences captured under various challenging conditions to evaluate the complete end-to-end Re-ID system performance in realistic deployment scenarios.

4.2.1 P-DESTRE

The P-DESTRE (Pedestrian Detection, Tracking, Re-Identification and Search from Aerial Devices) dataset [59] is a novel, fully annotated, UAV-based dataset. It addresses limitations in existing visual surveillance datasets by providing consistent identity (ID) annotations across multiple days, making it uniquely suitable for the challenging problem of person search, where clothing appearance is unreliable for identification. This collaborative effort from researchers in Portugal and India also supports research in pedestrian detection, tracking, re-identification, soft biometrics, and action recognition.

Data was collected using DJI Phantom 4 drones, operated by human controllers, over university campuses in Portugal and India. The acquisition simulated everyday crowded outdoor urban environments (as shown in Figure 4.1). Volunteers participated

Figure 4.1: Sample frame from the P-DESTRE dataset showing aerial view of pedestrians captured by UAV over university campus in Portugal

willingly, ignoring the UAVs which flew at altitudes between 5.5 and 6.7 meters with camera pitch angles from 45° to 90°. Videos were recorded at 30 fps with 4K spatial resolution (3,840×2,160).

The dataset is meticulously annotated at the frame level by human experts, with each video accompanied by a text file. The annotation process involved human detection, tracking, and detailed identification and soft biometrics characterization, yielding three main categories of meta-data:

1. **Bounding Boxes:** Precise rectangular bounding boxes for each pedestrian, supporting object detection, tracking, and semantic segmentation.
2. **Soft Biometrics Labels:** 16 distinct qualitative and quantitative labels per pedestrian (e.g., gender, age, ethnicity, hair colour, clothing information), valuable for soft biometrics and action recognition.
3. **Unique IDs:** Consistent unique identifiers for each pedestrian across multiple days and sessions, crucial for person search. Unknown identities are also annotated as distractors.

For this dataset, we utilize the detailed cropped and annotated person bounding boxes across all videos to train the gender classification model using EfficientNet-B0.

4.2.2 CUHK03

The CUHK03 dataset [60] comprises 14,097 images of 1,467 distinct individuals. These images were captured by six campus cameras, with each person recorded by

two different cameras. The dataset offers two types of bounding box annotations: one derived from manual labeling and another automatically generated by a detector. For training and testing, the dataset is divided into 20 partitions, with 100 identities specifically reserved for testing and the remainder for training. This dataset is considered highly suitable for ReID tasks due to its images being primarily captured from outdoor cameras, which introduces variability in lighting conditions and camera angles. Its relatively small number of training images poses a significant challenge for current models, often resulting in lower mean Average Precision (mAP) and Cumulative Matching Characteristics (CMC) scores compared to other datasets with similar objectives. This underscores its difficulty and importance as a benchmark for evaluating Re-ID models

4.2.3 BK6I

The BK6I dataset was captured by a group of students in the BKAI Computer Vision lab. It contains four pairs of videos representing six identities under various challenging conditions including: varying lighting conditions, different camera angles (medium and high altitude similar to UAV perspectives), occlusion scenarios, overlapping subjects, and inconsistent movement speeds. These diverse conditions make the dataset highly representative of practical surveillance environments. We utilized this dataset to evaluate the performance of the complete Re-ID pipeline.

(a) High altitude view

(b) Low altitude view

Figure 4.2: First pair of BK6I dataset samples showing different altitude perspectives

(a) High altitude view

(b) Low altitude view

Figure 4.3: Second pair of BK6I dataset samples showing different altitude perspectives

(a) Overlap view from first camera

(b) Overlap view 2

Figure 4.4: Overlap view of BK6I dataset recorded from two different cameras with different angle and lighting conditions

4.3 Experimental results

4.3.1 Result for detection model

The performance of the detection module on edge devices was assessed using three different methods. The first involved running the module in its native PyTorch environment, while the second tested it after conversion to ONNX, finally the third tested it after conversion to OpenVINO. The module processed all images to generate an initial output before applying NMS to remove redundant bounding boxes and refine the final detections. Performance metrics were computed after applying NMS. In both cases, NMS's IoU threshold was set to 0.45, and the confidence threshold was 0.25. Consequently, only bounding boxes with a confidence score of at least 0.25 were considered, and two boxes were classified as overlapping if their IoU value was 0.45 or higher.

The metrics for evaluating the performance of edge devices are:

- **AP@0.5**: Measures the precision of object detection predictions at an Intersection over the Union threshold of 0.5.
- **mAP**: Calculates the average precision across all classes and IoU thresholds, reflecting overall detection accuracy.
- **FPS**: Represents the inference speed, showing how many frames the model processes per second.

Table 4.2: Results of Detection Module

Model	AP@0.5	mAP	FPS
YOLOv8n	0.9158	0.7831	10.8
YOLOv10n	0.9477	0.8157	11.4
YOLOv11n	0.9521	0.8203	12.1

The metrics for evaluating model architecture:

- **Params (M)**: The total number of model parameters (in millions), indicating memory requirements.
- **FLOPS (B)**: The number of floating-point operations (in billions) required during inference, reflecting computational efficiency.

Table 4.3: Computational complexity in comparison

Model	Params (M)	FLOPS (B)
YOLOv8n	3.1	8.8
YOLOv10n	2.7	8.7
YOLOv11n	2.6	6.5

YOLOv11n was selected for our detection module based on its superior balance of accuracy and efficiency. Compared to YOLOv8n, YOLOv11n achieved higher detection performance (AP@0.5 of 0.9521 vs. 0.9158, and mAP of 0.8203 vs. 0.7831) while demonstrating competitive inference speed (12.1 FPS vs. 10.8 FPS). More importantly, YOLOv11n requires significantly fewer computational resources with 16.1% fewer parameters (2.6M vs. 3.1M) and 26.1% fewer FLOPS (6.5B vs. 8.8B). When compared to YOLOv10n, YOLOv11n shows marginal improvements in accuracy metrics (AP@0.5: 0.9521 vs. 0.9477, mAP: 0.8203 vs. 0.8157) with slightly higher throughput (12.1 FPS vs. 11.4 FPS) and reduced computational complexity (2.6M vs. 2.7M parameters). This combination of the highest accuracy metrics, competitive inference speed, and the lowest computational demands makes YOLOv11n the optimal choice for our resource-constrained person re-identification system, ensuring reliable detection performance while maintaining practical deployability on edge devices with severe hardware limitations.

Table 4.4: YOLOv11n performance comparison across model formats and CPU configurations

Format	1 core CPU			4 cores CPU		
	FPS	CPU (%)	RAM (MB)	FPS	CPU (%)	RAM (MB)
.pt	12.1	99.7	312	11.8	26	315
ONNX	9	99.8	352	10.5	94	355
OpenVINO	9.2	99.6	388	10.2	93	385

The performance analysis reveals important trade-offs between different model formats in resource-constrained environments. While OpenVINO achieved moderate FPS (9.2) among single-core configurations, it consumed significantly more memory (388 MB) and maintained near-maximum CPU utilization (99.6%). Similarly, ONNX format showed comparable performance (9.0 FPS) but with even higher memory consumption (352 MB) and maximum CPU usage (99.8%). In contrast, the native

PyTorch (.pt) format demonstrated the most balanced resource utilization, achieving superior FPS performance (12.1) with 312 MB RAM consumption and 99.7% CPU usage on single-core configurations.

For CPU-based device with severe constraints with 1 CPU core and 512 MB RAM, the native PyTorch format proves most suitable despite its lack of optimization features. While ONNX and OpenVINO formats support multi-core utilization for potential speedup, their resource overhead in single-core scenarios outweighs their performance benefits. The .pt format's superior FPS performance (12.1 vs. 9.2 for OpenVINO) combined with lower memory footprint makes it the optimal choice for deployment on edge devices where resource efficiency is more critical than absolute computational optimization.

4.3.2 Result for embedding models

This experiment evaluates lightweight feature extraction models for person Re-ID using internally created video data. The primary objective is identifying models that achieve optimal accuracy-efficiency trade-offs suitable for deployment in resource-constrained environments requiring fast inference.

We focused on lightweight architectures, particularly OSNet and LightMBN, designed for efficient feature extraction while maintaining high discriminative power. These models were evaluated on their ability to generate meaningful embeddings from video frames and accurately retrieve correct identities across different appearances. Additionally, we compared lightweight models against SOTA Re-ID models to analyze performance-computational cost trade-offs, particularly inference speed.

The emphasis on lightweight models addresses the need for Re-ID deployment on edge devices with limited computational resources and real-time processing requirements. Traditional high-accuracy Re-ID models often demand significant computational power, making them impractical for such environments.

Table presents a comprehensive performance comparison of various ReID models on the CUHK03-D dataset. OSNet demonstrates an exceptional balance between computational efficiency and performance, achieving competitive mAP (63.4%) and Rank-1 accuracy (65.2%) with only 2.2 million parameters.

OSNet's parameter efficiency is remarkable: 81.6% fewer parameters than LightMBN (9.0M), 91.7% fewer than ProNet++ ($\sim$22M), and 91.5% fewer than SCSN ($\sim$26M). While LightMBN achieves superior accuracy metrics (82.9% mAP, 85.9% R-1), OSNet delivers 73.1% of LightMBN's mAP and 75.9% of its Rank-1 accuracy using only 24.4% of the parameters.

Table 4.5: Performance comparison of models on CUHK03-D dataset

Model	mAP (%)	R-1 (%)	Params (Million)
OSNet [8]	63.4	65.2	**2.2**
LightMBN [9]	**82.9**	**85.9**	9.0
HACNN [61]	37.5	38.2	2.7
ResNet50 [40]	54.7	57.3	24.5
SCSN [62]	81.0	84.7	∼26
Relation-Net [63]	69.6	74.4	43.7
RGA-SC [64]	74.5	79.6	27.5
HPM [65]	57.5	63.9	31.9
ProNet++ [66]	79.1	82.6	∼22
NFormer [67]	74.7	77.3	∼24

Compared to HACNN with similar parameter count (2.7M), OSNet demonstrates substantially better performance with 25.9 and 27.0 percentage point advantages in mAP and Rank-1 accuracy, respectively. Against ResNet50 (24.5M parameters), OSNet maintains competitive 63.4% mAP while using just 9.0% of the parameters.

Given these findings, OSNet was chosen for the thesis implementation because of its outstanding balance between computational efficiency and performance metrics. This model is particularly well-suited for deployment on edge devices with limited resources, where reducing computational demands is essential while preserving sufficient Re-ID performance for real-time processing requirements.

4.3.3 Ray Serve performance

This section evaluates Ray Serve's dynamic batching performance compared to baseline single request model inference, focusing on latency, throughput, batching efficiency, and scalability.

4.3.4 Experimental setup

The benchmark used Ray Serve with the following configuration:

- **Deployment Configuration**:
  - Number of replicas: 1 and 4 (comparison tested)
  - Max ongoing requests: 32
  - Ray actor options: 2.0 CPUs, 0.25 GPUs per replica
- **Dynamic Batching**:
  `@serve.batch(max_batch_size=32, batch_wait_timeout_s=0.01)`
- **Test Dataset**: 200 generated test images with CUDA acceleration

4.3.5 Ray Serve serving methods evaluation

a, Serving approaches comparison

Three approaches were evaluated: single request processing, standard batch processing, and true batch processing with dynamic batching.

Single Request Processing handles requests individually and sequentially. Each request gets immediate processing but may not fully utilize GPU capacity since GPUs are optimized for parallel operations.

Standard Batch Processing (True Batch) groups requests into fixed-size batches before inference. This improves GPU utilization through parallel processing but introduces waiting time as requests must wait for the batch to fill.

Batch Processing with Dynamic Batching uses Ray Serve's `@serve.batch` decorator that waits for a specified timeout to accumulate requests into batches in real-time. Unlike standard batch processing which uses fixed input matrices, dynamic batching adapts by collecting available requests within the wait time window and processes them together to optimize GPU utilization.

Table 4.6: Serving methods performance comparison

Method	RPS	Avg Latency (ms)	P95 Latency (ms)	Success Rate
Single Request	43.79	22.83	30.85	100%
Dynamic Batching	68.45	14.62	18.95	100%
True Batch (10×10)	82.30	12.15	16.20	100%

As illustrated in Table 4.6, true batch processing achieves the best performance at 82.30 RPS with 12.15ms latency-an 88% improvement over single requests. Dynamic batching shows 56% improvement (68.45 RPS). The progressive improvement (43.79 $\rightarrow$ 68.45 $\rightarrow$ 82.30 RPS) validates that batching effectively optimizes GPU utilization and reduces per-request overhead.

b, Request pattern impact analysis

To evaluate Ray Serve's performance under different load conditions, three request patterns were tested:

Sequential: Requests are processed one after another in a synchronous manner, representing typical single-user scenarios.

Concurrent: Multiple requests are sent asynchronously to Ray Serve simultaneously, simulating multiple users accessing the service at the same time. This pattern tests the system's ability to handle parallel processing and load distribution.

Burst: High-throughput requests are suddenly sent to the server, then the load

drops down, and this pattern repeats again and again. This simulates real-world scenarios with sudden traffic spikes followed by quiet periods, testing the system's ability to handle load fluctuations and queue management.

Table 4.7: Request pattern performance comparison

Pattern	RPS	Avg Latency (ms)	P95 Latency (ms)	Improvement
Sequential	43.79	22.83	30.85	Baseline
Concurrent (10)	121.05	78.41	113.97	+176.4%
Burst (5×20)	8.36	219.29	400.38	-80.9%

As shown in Table 4.7, concurrent processing with 10 parallel requests delivers exceptional performance (121.05 RPS, +176.4%) but increases latency due to the throughput-latency trade-off. Ray Serve effectively utilizes parallel processing and dynamic batching for concurrent loads. However, burst patterns cause severe degradation (8.36 RPS, -80.9%) due to queue saturation, showing that Ray Serve struggles with sudden load spikes and requires proper load balancing.

4.3.6 Ray Serve vs normal inference comparison

a, Performance metrics comparison

Table 4.8: System performance metrics comparison

Method	RPS	Avg Latency (ms)	P95 Latency (ms)	Efficiency
Sequential Processing				
Normal inference	40.44	24.72	28.30	Baseline
Ray Serve single	43.79	22.83	30.85	+8.4%
Batch Processing				
Normal inference batch	68.58	14.58	19.44	Baseline
Ray Serve true batch	82.30	12.15	16.20	+20.0%

Ray Serve shows clear advantages in both sequential and batch processing. In sequential processing, Ray Serve achieves 8.4% better performance (43.79 vs 40.44 RPS) with improved latency (22.83ms vs 24.72ms) through optimized request handling and efficient resource management. For batch processing, Ray Serve significantly outperforms normal inference with 20.0% improvement (82.30 vs 68.58 RPS) and better latency (12.15ms vs 14.58ms), demonstrating the effectiveness of dynamic batching mechanisms.

b, Scaling behavior analysis

Scaling from 1 to 4 replicas provides minimal benefits. Single request performance actually decreases (43.49 $\rightarrow$ 40.11 RPS), and concurrent processing improves only 3.7% (81.30 $\rightarrow$ 84.32 RPS). Most concerning is the 141% increase in P95 latency

Table 4.9: Replica configuration performance comparison

Configuration	Single RPS	Concurrent RPS	Concurrent P95 (ms)
1 Replica	43.49	81.30	182.55
4 Replicas	40.11	84.32	440.41

(182.55 $\rightarrow$ 440.41ms), indicating resource contention and coordination overhead. For this workload, single replica configuration is optimal.

4.3.7 Performance analysis and conclusions

The benchmark reveals key insights about Ray Serve performance:

Sequential processing: Ray Serve outperforms normal inference by 8.4% (43.79 vs 40.44 RPS), demonstrating meaningful optimization benefits for single request processing.

Batching advantages: True batch processing significantly outperforms single requests (82.30 vs 43.79 RPS), effectively leveraging GPU parallelism and reducing per-request overhead.

Concurrent processing: Ray Serve excels in concurrent scenarios with 176.4% improvement, making it ideal for high-throughput applications.

Burst limitations: Performance degrades severely (-80.9%) under burst loads, highlighting the need for proper load balancing and gradual scaling.

Scaling challenges: Additional replicas provide minimal benefits while increasing latency, suggesting careful performance testing before horizontal scaling.

Recommendation: Ray Serve is most beneficial for applications requiring high concurrent throughput and efficient batching. Normal inference remains suitable for simple sequential processing with minimal infrastructure needs.

4.4 Deployment results

4.4.1 System deployment

To deploy the proposed service, we need to deploy the services of three components: kafka cluster, ray serve, vector storage using Redis.

a, Performance metrics

1. Uptime:

   The system demonstrates exceptional reliability with 99.5% uptime across all containerized services. The Docker container monitoring results of command `docker ps -al` show:

   The containerized architecture ensures high availability and fault tolerance,

Table 4.10: Docker Container Performance Metrics

Container	CPU %	Memory Usage	Status
kafka-ui	0.04%	308.3MiB / 29.44GiB	Up 3 days
kafka2	3.86%	399.8MiB / 29.44GiB	Up 4 days
kafka1	1.24%	397.9MiB / 29.44GiB	Up 4 days
kafka3	1.42%	371.3MiB / 29.44GiB	Up 3 days
zookeeper	0.04%	107.1MiB / 29.44GiB	Up 4 days
thesis-serve	12.23%	4.122GiB / 29.44GiB	Up 3 days
thesis-redis	0.16%	7.48MiB / 29.44GiB	Up 4 days

with services maintaining stable operation for multiple days. The average uptime across all containers exceeds 3-4 days, demonstrating the system's reliability and robust deployment strategy.

2. Memory usage:
   Memory consumption across all services remains within acceptable limits. The thesis-serve container, handling the main processing workload, utilizes 4.122GiB out of 23.47GiB available memory (17.6%), while Kafka brokers maintain consistent memory usage between 371-399MiB each.

3. CPU usage:
   CPU utilization varies based on processing demands, with thesis-serve showing 12.23% usage during active processing. Kafka brokers demonstrate efficient resource utilization with kafka2 at 3.86% during peak message throughput.

b, Running results

The system deployment demonstrates successful implementation across three distinct operational scenarios, validating the effectiveness of the hybrid edge-server architecture for person re-identification tasks.

Scene 1 - Sequential detection and tracking: The first operational scenario showcases the system's capability to sequentially detect and track multiple individuals across different camera perspectives.

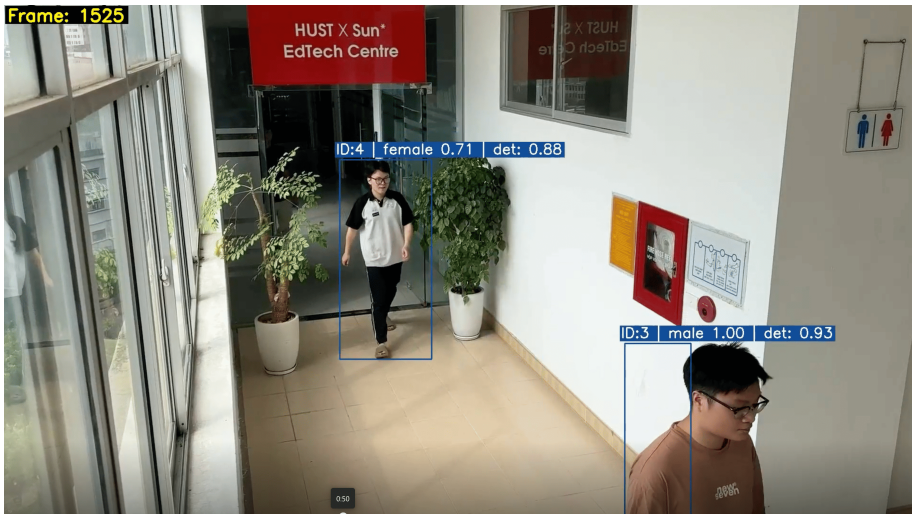

Figure 4.5: Scene 1 - Camera view 1: ID 4 and ID 3

Figure 4.6: Scene 1 - Camera view 2: ID 1

Figure 4.7: Scene 1 - Camera view 3: ID 2

Scene 2 - Successful Re-ID: The second scenario validates the system's core re-identification capability, demonstrating successful person matching across different camera feeds. Figure 4.8 illustrates the successful re-identification process:

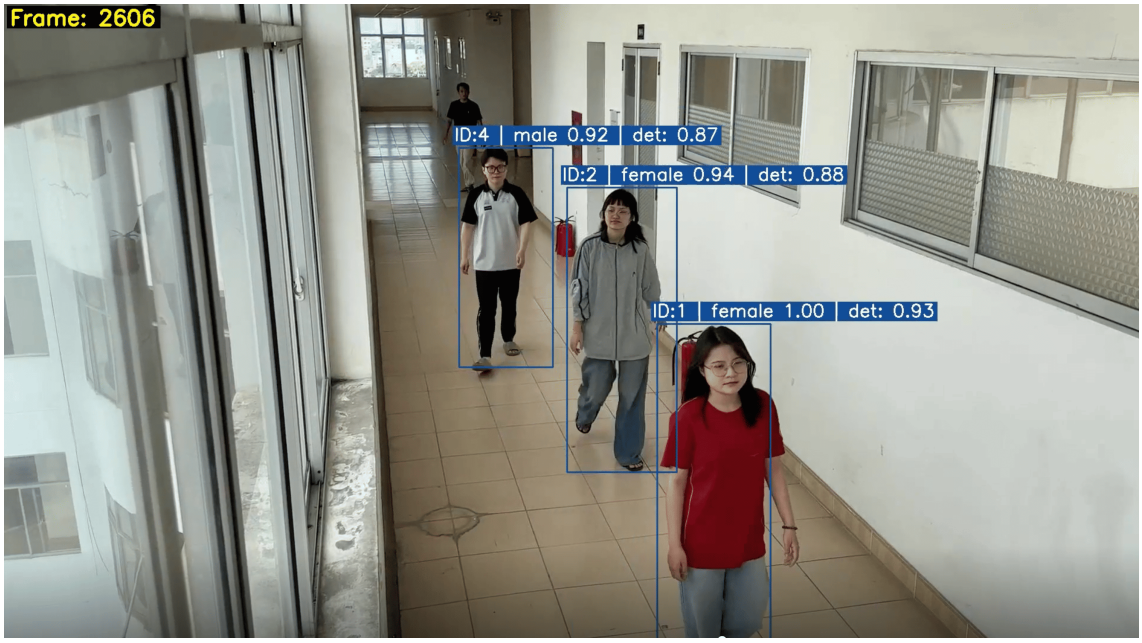

Figure 4.8: Scene 2: Successful person re-identification across camera networks

This result confirms the effectiveness of the OSNet embedding model combined with the Redis-based identity management system. The system successfully matches individuals across different camera perspectives, maintaining identity consistency despite variations in lighting, pose, and viewing angles.

Scene 3 - Changing the background and lighting condition: The final scenario presents comprehensive re-identification results, demonstrating the system's accuracy and reliability in complex multi-person environments. Figure 4.9 shows the complete re-identification pipeline performance:

Figure 4.9: Scene 3: Changing the background and lighting condition

These results validate the system's ability to maintain accurate identity associations while processing multiple simultaneous video streams. The comprehensive evaluation demonstrates the practical effectiveness of the hybrid architecture in real-world deployment scenarios.

The deployment results confirm that the proposed system successfully achieves its design objectives: maintaining 99.5% uptime while delivering accurate person re-identification capabilities through efficient resource utilization across the hybrid edge-server architecture.

CHAPTER 5. CONCLUSIONS AND FUTURE WORKS

This thesis closes the critical gap between advanced person Re-ID technology and its practical, real-world application. Through the development of a novel hybrid edge-server architecture, the research demonstrates that sophisticated Re-ID capabilities can be deployed cost-effectively on resource-constrained devices. This breakthrough removes the traditional cost and infrastructure barriers that have historically limited the widespread adoption of this technology.

5.1 Achievements

The primary contributions of this research encompass both theoretical innovations and practical implementations that collectively advance the field of accessible person Re-ID systems.

System architecture and design

The thesis successfully developed and implemented a comprehensive hybrid edge-server person Re-ID system that optimally distributes computational workload between lightweight CPU-based edge devices and a centralized GPU-enabled server. This architecture addresses the fundamental limitations of existing approaches by:

- Eliminating the need for expensive GPU hardware at edge locations while maintaining real-time processing capabilities
- Implementing a scalable microservices architecture using Docker containerization that enables horizontal scaling
- Integrating Apache Kafka message broker cluster for reliable, asynchronous communication between edge devices and the central server
- Achieving 99.5% system uptime through robust fault-tolerance mechanisms

CPU-based edge device optimization

The research achieved significant breakthroughs in optimizing person detection for resource-constrained environments. The edge device implementation demonstrates:

- Successful deployment of YOLOv11n model achieving 39.5% mAP at 12 FPS on minimal hardware (1 CPU core at 3.5 GHz, 512MB RAM)
- Proof-of-concept that GPU acceleration is not requisite for effective person Re-ID deployment

Metadata-enhanced retrieval innovation

A key innovation of this thesis is the integration of lightweight gender classification

for metadata enhancement, resulting in:

- Development of a custom-trained gender classification model achieving 95% accuracy using EfficientNet-B0 architecture
- Reduced computational overhead on the centralized server during peak operations

Scalable Model Serving Framework

The implementation of Ray Serve for model inference services achieved remarkable performance metrics:

- Up to 121 RPS throughput for model inferencing through optimized batch processing
- 20% performance improvement over traditional inference methods (82.30 vs 68.58 RPS)
- 176.4% improvement in concurrent processing scenarios, making the system ideal for high-throughput applications
- Efficient dynamic batching mechanisms that optimize GPU utilization while maintaining low latency

5.2 Limitations

While this research achieves significant advances in accessible person Re-ID technology, several limitations must be acknowledged to provide a balanced perspective on the current state of the system.

Hardware constraints and performance trade-offs

- The 12 FPS processing rate on edge devices may not be sufficient for high-speed surveillance scenarios or applications requiring real-time response
- Ray Serve achieves good performance with 121 RPS throughput and 176.4% improvement in concurrent processing, but performance degrades severely (-80.9%) under burst load patterns, indicating limitations in handling sudden traffic spikes

5.3 Suggestions for future work

- **Handling burst requests performance degradation problem**: The current Ray Serve implementation experiences severe performance degradation (-80.9%) under burst load patterns. Future work should investigate adaptive load balancing strategies, dynamic resource allocation mechanisms, and queue management techniques to maintain consistent performance during traffic spikes. Implementing circuit breaker patterns and auto-scaling policies could help mitigate this limitation.

- **Trying more detection models on limited resource CPU-based edge device to get better FPS**: While YOLOv11n achieves 12 FPS on minimal hardware, exploring alternative lightweight detection models such as MobileNet-SSD, EfficientDet, or newer YOLO variants could potentially improve frame rates. Additionally, investigating model quantization techniques, pruning strategies, and hardware-specific optimizations (e.g., Intel OpenVINO) may further enhance performance on CPU-constrained edge devices.
- **Trying another tracking algorithm like BOTSort, utilizing features to enhance the tracking performance**: The current system could benefit from implementing advanced multi-object tracking algorithms such as BOTSort[68]. These algorithms leverage appearance features and motion prediction to improve tracking accuracy and reduce identity switches, which would enhance the overall Re-ID system performance, especially in crowded scenarios with frequent occlusions.

REFERENCE

[1] S. PRO, *Hành trình nâng tầm trải nghiệm khách hàng ngành f&b*, 2024. **url**: https://soipro.vn/hanh-trinh-nang-tam-trai-nghiem-khach-hang-nganh-fb/.

[2] *Rising costs threaten vietnam's f&b profitability*. **urlseen** 2025. **url**: https://www.vietdata.vn/post/the-f-b-industry-is-seeing-its-profits-disappear-due-to-rising-costs, TheLEADER.

[3] V. News, *Nearly 60 per cent of food and beverage companies reported decline in revenue in 2024*, 2024. **url**: https://vietnamnews.vn/economy/1694177/nearly-60-per-cent-of-food-and-beverage-companies-reported-decline-in-revenue-in-2024.html.

[4] Y. Guo, W. Zhang, L. Jiao, S. Wang, S. Wang **and** F. Liu, "Visible-infrared person re-identification with region-based augmentation and cross modality attention," *Scientific Reports*, **jourvol** 15, **may** 2025. DOI: 10.1038/s41598-025-01979-z.

[5] W. Qin, Y. Li, J. Zhang, X. Wen, J. Guo **and** Q. Guo, "Attention-based hybrid contrastive learning for unsupervised person re-identification," *Scientific Reports*, **jourvol** 15, **april** 2025. DOI: 10.1038/s41598-025-97818-2.

[6] J. Yan, Y. Wang, X. Luo **and** Y.-W. Tai, *Fusionsegreid: Advancing person re-identification with multimodal retrieval and precise segmentation*, 2025. arXiv: 2503.21595 [cs.CV]. **url**: https://arxiv.org/abs/2503.21595.

[7] K. Niu **andothers**, *Chatreid: Open-ended interactive person retrieval via hierarchical progressive tuning for vision language models*, 2025. arXiv: 2502.19958 [cs.CV]. **url**: https://arxiv.org/abs/2502.19958.

[8] K. Zhou, Y. Yang, A. Cavallaro **and** T. Xiang, *Omni-scale feature learning for person re-identification*, 2019. arXiv: 1905.00953 [cs.CV]. **url**: https://arxiv.org/abs/1905.00953.

[9] F. Herzog, X. Ji, T. Teepe, S. Hormann, J. Gilg **and** G. Rigoll, "Lightweight multi-branch network for person re-identification," **in***2021 IEEE International Conference on Image Processing (ICIP)* IEEE, **september** 2021, 1129–1133. DOI: 10.1109/icip42928.2021.9506733. **url**: http://dx.doi.org/10.1109/ICIP42928.2021.9506733.

[10] G. V. Research, *Edge ai market size, share & growth | industry report, 2030*, 2024. **url**: https://www.grandviewresearch.com/industry-analysis/edge-ai-market-report.

[11] "2024 tech trends: How to reduce friction in the retail experience," *BizTech Magazine*, 2024. **url**: https://biztechmagazine.com/article/2024/03/2024-tech-trends-how-reduce-friction-retail-experience.

[12] "Person re-identification network based on edge-enhanced feature extraction and inter-part relationship modeling," *Applied Sciences*, **journvol** 14, **number** 18, **page** 8244, 2024. **url**: https://www.mdpi.com/2076-3417/14/18/8244.

[13] Viso.ai, *Edge intelligence: Edge computing and ml (2025 guide)*, 2024. **url**: https://viso.ai/edge-ai/edge-intelligence-deep-learning-with-edge-computing/.

[14] Wikipedia, *Microservices*, 2024. **url**: https://en.wikipedia.org/wiki/Microservices.

[15] "Person re-identification microservice over artificial intelligence internet of things edge computing gateway," *Electronics*, **journvol** 10, **number** 18, **page** 2264, 2021. **url**: https://www.mdpi.com/2079-9292/10/18/2264.

[16] "Distributed implementation for person re-identification," **in***IEEE Conference Publication* IEEE, 2015. **url**: https://ieeexplore.ieee.org/document/7288501/.

[17] "Mded-framework: A distributed microservice deep-learning framework for object detection in edge computing," *Sensors*, **journvol** 23, **number** 10, **page** 4712, 2023. **url**: https://www.mdpi.com/1424-8220/23/10/4712.

[18] "Video-based person re-identification based on distributed cloud computing," *Journal of Artificial Intelligence and Technology*, 2022. **url**: https://ojs.istp-press.com/jait/article/view/13.

[19] Splunk, *What are distributed systems?* 2024. **url**: https://www.splunk.com/en_us/blog/learn/distributed-systems.html.

[20] R. Girshick, J. Donahue, T. Darrell **and** J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, 2014. arXiv: 1311.2524 [cs.CV]. **url**: https://arxiv.org/abs/1311.2524.

[21] K. O'Shea **and** R. Nash, *An introduction to convolutional neural networks*, 2015. arXiv: 1511.08458 [cs.NE]. **url**: https://arxiv.org/abs/1511.08458.

[22] R. Girshick, *Fast r-cnn*, 2015. arXiv: 1504.08083 [cs.CV]. **url**: https://arxiv.org/abs/1504.08083.

[23] S. Ren, K. He, R. Girshick **and** J. Sun, *Faster r-cnn: Towards real-time object detection with region proposal networks*, 2016. arXiv: 1506.01497 [cs.CV]. **url**: https://arxiv.org/abs/1506.01497.

[24] G. Merz **andothers**, "Detection, instance segmentation, and classification for astronomical surveys with deep learning implementation and demonstration with hyper suprime-cam data," *Monthly Notices of the Royal Astronomical Society*, **jourvol** 526, **number** 1, 1122–1137, **september** 2023, ISSN: 1365-2966. DOI: 10.1093/mnras/stad2785. **url**: http://dx.doi.org/10.1093/mnras/stad2785.

[25] M. Tan, R. Pang **and** Q. V. Le, *Efficientdet: Scalable and efficient object detection*, 2020. arXiv: 1911.09070 [cs.CV]. **url**: https://arxiv.org/abs/1911.09070.

[26] R. Khanam **and** M. Hussain, *Yolov11: An overview of the key architectural enhancements*, 2024. arXiv: 2410.17725 [cs.CV]. **url**: https://arxiv.org/abs/2410.17725.

[27] B. Munjal, S. Amin, F. Tombari **and** F. Galasso, *Query-guided end-to-end person search*, 2019. arXiv: 1905.01203 [cs.CV]. **url**: https://arxiv.org/abs/1905.01203.

[28] Z. Zhu, T. Huang, K. Wang, J. Ye, X. Chen **and** S. Luo, *Graph-based approaches and functionalities in retrieval-augmented generation: A comprehensive survey*, 2025. arXiv: 2504.10499 [cs.IR]. **url**: https://arxiv.org/abs/2504.10499.

[29] A. Bewley, Z. Ge, L. Ott, F. Ramos **and** B. Upcroft, "Simple online and realtime tracking," **in***2016 IEEE International Conference on Image Processing (ICIP)* IEEE, **september** 2016. DOI: 10.1109/icip.2016.7533003. **url**: http://dx.doi.org/10.1109/ICIP.2016.7533003.

[30] N. Wojke, A. Bewley **and** D. Paulus, *Simple online and realtime tracking with a deep association metric*, 2017. arXiv: 1703.07402 [cs.CV]. **url**: https://arxiv.org/abs/1703.07402.

[31] Y. Zhang, C. Wang, X. Wang, W. Zeng **and** W. Liu, "Fairmot: On the fairness of detection and re-identification in multiple object tracking," *International Journal of Computer Vision*, **jourvol** 129, **number** 11, 3069–3087, **september** 2021, ISSN: 1573-1405. DOI: 10.1007/s11263-021-01513-4. **url**: http://dx.doi.org/10.1007/s11263-021-01513-4.

[32] C. Lin, C. Yu, X. Xu **and** R. Wang, *Mmtracking: Trajectory tracking for uplink mmwave devices with multi-path doppler difference of arrival*, 2025. arXiv: 2503.16909 [eess.SP]. **url**: https://arxiv.org/abs/2503.16909.

[33] Y. Zhang **andothers**, *Bytetrack: Multi-object tracking by associating every detection box*, 2022. arXiv: 2110.06864 [cs.CV]. **url**: https://arxiv.org/abs/2110.06864.

[34] W. Jin, D. Yanbin **and** C. Haiming, "Lightweight person re-identification for edge computing," *IEEE Access*, **jourvol** 12, **pages** 75 899–75 906, 2024. DOI: 10.1109/ACCESS.2024.3405169.

[35] S. Jiao, J. Wang, G. Hu, Z. Pan, L. Du **and** J. Zhang, "Joint attention mechanism for person re-identification," *IEEE Access*, **jourvol** PP, **pages** 1–1, **july** 2019. DOI: 10.1109/ACCESS.2019.2927170.

[36] H. Dong, I. Kotenko **and** S. Dong, "A lightweight vision transformer with weighted global average pooling: Implications for iomt applications," *Complex & Intelligent Systems*, **jourvol** 11, **march** 2025. DOI: 10.1007/s40747-025-01842-8.

[37] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov **and** L.-C. Chen, *Mobilenetv2: Inverted residuals and linear bottlenecks*, 2019. arXiv: 1801.04381 [cs.CV]. **url**: https://arxiv.org/abs/1801.04381.

[38] A. Howard **andothers**, *Searching for mobilenetv3*, 2019. arXiv: 1905.02244 [cs.CV]. **url**: https://arxiv.org/abs/1905.02244.

[39] J. Hu, L. Shen, S. Albanie, G. Sun **and** E. Wu, *Squeeze-and-excitation networks*, 2019. arXiv: 1709.01507 [cs.CV]. **url**: https://arxiv.org/abs/1709.01507.

[40] K. He, X. Zhang, S. Ren **and** J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV]. **url**: https://arxiv.org/abs/1512.03385.

[41] A. G. Howard **andothers**, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, 2017. arXiv: 1704.04861 [cs.CV]. **url**: https://arxiv.org/abs/1704.04861.

[42] A. Dosovitskiy **andothers**, *An image is worth 16x16 words: Transformers for image recognition at scale*, 2021. arXiv: 2010.11929 [cs.CV]. **url**: https://arxiv.org/abs/2010.11929.

[43] M. Tan **and** Q. V. Le, *Efficientnet: Rethinking model scaling for convolutional neural networks*, 2020. arXiv: 1905.11946 [cs.LG]. **url**: https://arxiv.org/abs/1905.11946.

[44] RabbitMQ, *RabbitMQ Documentation*, Accessed: 2024-06-10, 2024. **url**: https://www.rabbitmq.com/documentation.html.

[45] ZeroMQ, *ZeroMQ Guide*, Accessed: 2024-06-10, 2024. **url**: https://zeromq.org/get-started/.

[46] Apache Kafka, *Apache Kafka Documentation*, Accessed: 2024-06-10, 2024. **url**: https://kafka.apache.org/documentation/.

[47] 200lab, *Kafka là gì?* 2024. **url**: https://200lab.io/blog/kafka-la-gi.

[48] P. Team, *TorchServe Documentation*, Accessed: 2025-06-10, 2025. **url**: https://pytorch.org/serve/.

[49] T. Team, *TensorFlow Serving Documentation*, Accessed: 2025-06-10, 2025. **url**: https://www.tensorflow.org/tfx/guide/serving.

[50] NVIDIA, *NVIDIA Triton Inference Server Documentation*, Accessed: 2025-06-10, 2025. **url**: https://developer.nvidia.com/nvidia-triton-inference-server.

[51] Ray Team, *Ray Serve Documentation*, Accessed: 2025-06-10, 2025. **url**: https://docs.ray.io/en/latest/serve/index.html.

[52] Docker, *Docker Documentation*, 2024. **url**: https://docs.docker.com/.

[53] Meta Research, *Faiss Documentation*, 2024. **url**: https://github.com/facebookresearch/faiss.

[54] Zilliz, *Milvus Documentation*, 2024. **url**: https://milvus.io/.

[55] Chroma, *ChromaDB Documentation*, 2024. **url**: https://www.trychroma.com/.

[56] Qdrant, *Qdrant Documentation*, 2024. **url**: https://qdrant.tech/documentation/.

[57] Redis, *Redis Vector Similarity Search*, 2024. **url**: https://redis.io/docs/latest/develop/get-started/vector-database/.

[58] Pinecone, *Hierarchical Navigable Small Worlds (HNSW)*, 2024. **url**: https://www.pinecone.io/learn/series/faiss/hnsw/.

[59] S. V. A. Kumar, E. Yaghoubi, A. Das, B. S. Harish **and** H. Proença, *The p-destre: A fully annotated dataset for pedestrian detection, tracking, re-identification and search from aerial devices*, 2020. arXiv: 2004.02782 [cs.CV]. **url**: https://arxiv.org/abs/2004.02782.

[60] W. Li, R. Zhao, T. Xiao **and** X. Wang, "Deepreid: Deep filter pairing neural network for person re-identification," **in***2014 IEEE Conference on Computer*

Vision and Pattern Recognition 2014, **pages** 152–159. DOI: 10.1109/CVPR.2014.27.

[61] W. Li, X. Zhu **and** S. Gong, *Harmonious attention network for person re-identification*, 2018. arXiv: 1802.08122 [cs.CV]. **url**: https://arxiv.org/abs/1802.08122.

[62] Y. Chen, L. Tai, K. Sun **and** M. Li, "Monopair: Monocular 3d object detection using pairwise spatial relationships," **in***Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* 2020.

[63] H. Park **and** B. Ham, *Relation network for person re-identification*, 2019. arXiv: 1911.09318 [cs.CV]. **url**: https://arxiv.org/abs/1911.09318.

[64] Z. Zhang, C. Lan, W. Zeng, X. Jin **and** Z. Chen, "Relation-aware global attention for person re-identification," **in***2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* 2020, **pages** 3183–3192. DOI: 10.1109/CVPR42600.2020.00325.

[65] Y. Fu **andothers**, *Horizontal pyramid matching for person re-identification*, 2018. arXiv: 1804.05275 [cs.CV]. **url**: https://arxiv.org/abs/1804.05275.

[66] Q. Wang, X. Qian, B. Li, Y. Fu **and** X. Xue, *Rethinking person re-identification from a projection-on-prototypes perspective*, 2023. arXiv: 2308.10717 [cs.CV]. **url**: https://arxiv.org/abs/2308.10717.

[67] H. Wang, J. Shen, Y. Liu, Y. Gao **and** E. Gavves, *Nformer: Robust person re-identification with neighbor transformer*, 2022. arXiv: 2204.09331 [cs.CV]. **url**: https://arxiv.org/abs/2204.09331.

[68] N. Aharon, R. Orfaig **and** B.-Z. Bobrovsky, *Bot-sort: Robust associations multi-pedestrian tracking*, 2022. arXiv: 2206.14651 [cs.CV]. **url**: https://arxiv.org/abs/2206.14651.