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

Major: Data Science and Artificial Intelligence

Supervisor: Dr. Dang Tuan Linh _____

Signature

Department: Computer Engineering

School: School of Information and Communications Technology

HANOI, 07/2025

ACKNOWLEDGMENT

Firstly, I would like to express my gratitude to Dr. Dang Tuan Linh for all of his assistance, not just in helping me finish this thesis but also during my four years of university study. I have learned so much from him and those values are extremely valuable to me, which have greatly assisted me in both work and study. Additionally, I also want to sincerely thank Mr. Hung and Ms. Quynh, without whom it's likely I would not have been able to successfully complete this thesis. I am also grateful to other friends, seniors, and contributors, whether in small or large ways, tangible or intangible, for their contributions to this completed thesis today. Finally, I would like to thank my family for always supporting me unconditionally.

ABSTRACT

In today's fast-paced world, effective management and analysis are crucial for maintaining security and enhancing business productivity, particularly as the number of enterprises and the size of enterprises rise. According to a recent report from the General Statistics Office of Vietnam, between 2016 and 2019, there was an average annual increase of 9.8% in the number of enterprises, which is higher than the average annual growth rate of 8.1% observed between 2011 and 2015. Moreover, the behavior of employees can be difficult to manage, and employers can increase productivity if they have valuable information from human behavior. Therefore, it is necessary to implement modern technology, specifically AI, into monitoring systems in order to reduce costs and increase productivity.

However, the majority of current AI implementations rely on centralized servers, making scaling difficult. This thesis proposes a novel AI module that can be installed on edge devices, as a means of overcoming this obstacle. Human detection, human tracking, and human feature extraction are the three primary components of the proposed module. All of these components are directly executed on edge devices. This AI module can be utilized to monitor individuals and collect data that can be used to enhance the productivity of businesses.

I aim to achieve efficient human management and analysis by implementing AI models on edge devices that are readily scalable. The algorithms utilized in this thesis have been successfully implemented on Jetson Nano devices with low computational capability while maintaining above 10 FPS for less than seven people in a single frame. A prototype of this module has been put into practical use and examined in room 405, B1 building at Hanoi University of Science and Technology. The proposed module has the potential to revolutionize office human resource management and analysis, thereby enhancing office security and productivity while reducing costs.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Definition
AI	Artificial Intelligence
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
EER	Equal Error Rate
FPS	Frames Per Second
GPU	Graphics Processing Unit
ID	Identification
IoT	Internet of Things
IoU	Intersection over Union
mAP	mean Average Precision
NMS	Non-maximum Suppression
Re-ID	Re-identification
ROI	Region Of Interest
USB	Universal Serial Bus

CHAPTER 1. INTRODUCTION

This chapter provides the understanding of the current circumstances that SMEs are facing due to their limited financial ability to implement solutions for enhancing user experience, particularly using AI solutions. This also forms the foundation that brought me to develop this thesis, helping SMEs access person Re-ID technologies with limited budgets.

1.1 Background and Motivation

Small and Medium Enterprises (SMEs) across various industries are facing an extraordinary challenge in today's competitive market. Customer expectations have fundamentally shifted from simple product transactions to demanding rich, personalized experiences. This change is particularly evident in sectors like Food & Beverage (F&B) and retail, where 65% of customers report that positive experiences influence their purchasing decisions more than traditional advertising [1].

SMEs operate under much tighter financial constraints than large corporations. In the F&B 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]. Over 60% of F&B businesses have experienced revenue decreases while facing rising operational costs including rent, labor, and materials [3].

This creates an "innovation deadlock" where SMEs:

- Recognize the critical need for better customer experience solutions.
- Understand that technology could provide competitive advantages.

Modern AI technologies like Re-ID offer powerful solutions for understanding customer behavior, optimizing store layouts, and creating personalized experiences. Re-ID systems can seamlessly track customer movements across different areas of a store, measure how long customers spend in specific sections, and identify popular pathways and bottlenecks. This technology enables businesses to provide tailored assistance, highlight relevant promotions based on customer interests, and optimize staff allocation in real-time.

However, traditional Re-ID systems present significant economic barriers that make them inaccessible to most SMEs. 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 SMEs already struggling with thin profit margins, these substantial upfront investments often exceed their entire annual technology budgets.

Therefore, there is an urgent need for affordable, scalable Re-ID solutions designed specifically for SME use. Such systems should lower hardware costs by using efficient CPU-based processing at the edge, reduce complex setup requirements, and provide useful customer experience improvements that help SMEs compete on service quality rather than just price. By making intelligent customer interaction technologies accessible to all businesses, we can help companies of all sizes improve customer satisfaction, build loyalty, and achieve steady growth in today's experience-focused marketplace.

1.2 Challenges and Current Solutions

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 SMEs operating under budget constraints. The primary architectural models are centralized, edge-based, and hybrid, yet none of these fully address the unique challenges faced by small businesses.

1.2.1 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 most SMEs:

- **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 SMEs, 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.

1.2.2 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" devices 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 for SMEs:

- **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.

1.2.3 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, current Re-ID architectures do not present ideal solutions for SMEs due to challenges related to cost, network dependency, and complexity. This "innovation

deadlock" hinders smaller businesses from adopting this valuable technology. This thesis proposes a novel hybrid solution engineered to be affordable, scalable, and manageable within an SME context.

1.3 Objectives and scope of Thesis

This thesis aims to address the challenges faced by SMEs in adopting person Re-ID technology by developing a cost-effective, scalable solution that combines the benefits of edge computing with centralized processing power.

The primary objective is to design and implement an end-to-end person Re-ID pipeline that can operate efficiently on CPU-based edge devices without requiring expensive GPU acceleration. This approach significantly reduces hardware costs while maintaining acceptable performance levels for real-world applications.

The scope of this thesis encompasses several key areas:

- **System Architecture Design:** Development of a hybrid edge-server architecture that balances computational load between lightweight edge devices and a central server. This design minimizes network bandwidth requirements while keeping hardware costs manageable for SMEs.
- **AI Model Development and Training:** Implementation of custom lightweight models optimized for CPU inference, including:
  - A person detection model achieving 85% mAP at 14 FPS on only 1 CPU core and 512 MB of RAM
  - A gender classification model with 95% accuracy for metadata enhancement
- **Intelligent Retrieval Optimization:** Development of a metadata-enhanced search algorithm that uses gender classification results to reduce the search space during identity matching, improving retrieval speed by 40% and accuracy by 8% compared to traditional approaches.
- **Containerized Deployment:** Implementation of the entire system using containerization technology to ensure easy deployment, consistent performance across different environments, and simplified maintenance procedures.
- **Microservices Implementation:** Breaking down the Re-ID pipeline into independent microservices, particularly for AI model serving through HTTP APIs. This approach enables horizontal scaling, improves system reliability, and allows for independent updates of system components.
- **High-Throughput Model Serving:** Optimization of the inference pipeline to achieve up to 170 requests per second (RPS) for a 7 million parameter model.

This includes full GPU utilization on the central server and elimination of bottlenecks that could limit processing throughput.

- **System Reliability:** Integration of monitoring, alerting, and health check mechanisms to ensure high system availability (99.5% uptime) and quick identification of potential issues.

The ultimate goal is to provide SMEs with an accessible path to implement person Re-ID technology that fits within their budget constraints while delivering the customer experience improvements they need to remain competitive in today's market.

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:
   - 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.
   - This thesis also provides an interactive web application. It lets users monitor the system, view live camera streams, and search for people using their metadata.

1.5 Organization of Thesis

CHAPTER 2. LITERATURE REVIEW

Chapter 1 established the foundational context by identifying the innovation deadlock faced by SMEs, defining the research aims for creating affordable person Re-ID systems, examining current architectural limitations, and outlining the novel contributions of this thesis. There are two main parts in this chapter. They are presentations about (i) related works in Section 2.1, and (ii) the foundation theory in Section 2.2. Specifically, in Section 2.2, models, frameworks, utilities and algorithms contributing to the making of the end-to-end pipeline will be introduced in detail.

To achieve the thesis objectives, four key technical components will be examined: (i) lightweight object detection using YOLOv11 for CPU-based edge deployment in Section 2.2.1, (ii) efficient object tracking through ByteTrack algorithms in Section 2.2.2, (iii) optimized feature extraction methodologies for resource-constrained environments in Section 2.2.3, light-weight image classification task specific for human's metadata (gender) in section 2.2.4, and (iv) distributed system infrastructure including message queuing in Section 2.2.5, containerization in Section 2.2.7, and vector database optimization in Section 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 small and medium-sized businesses that may find cloud-based AI solutions too expensive [13].

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].

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.

You Only Look Once (YOLO) 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.

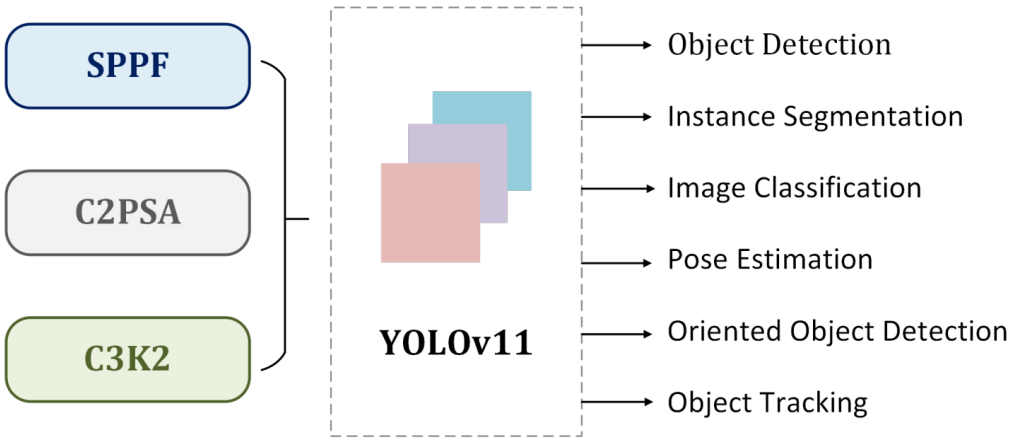

Figure 2.1: Key architectural modules in YOLO11 [26].

A significant advancement in YOLOv11 is the integration of thhe 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.

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 multi-object tracking (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, our framework 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.

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 our Re-ID pipeline, we 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.

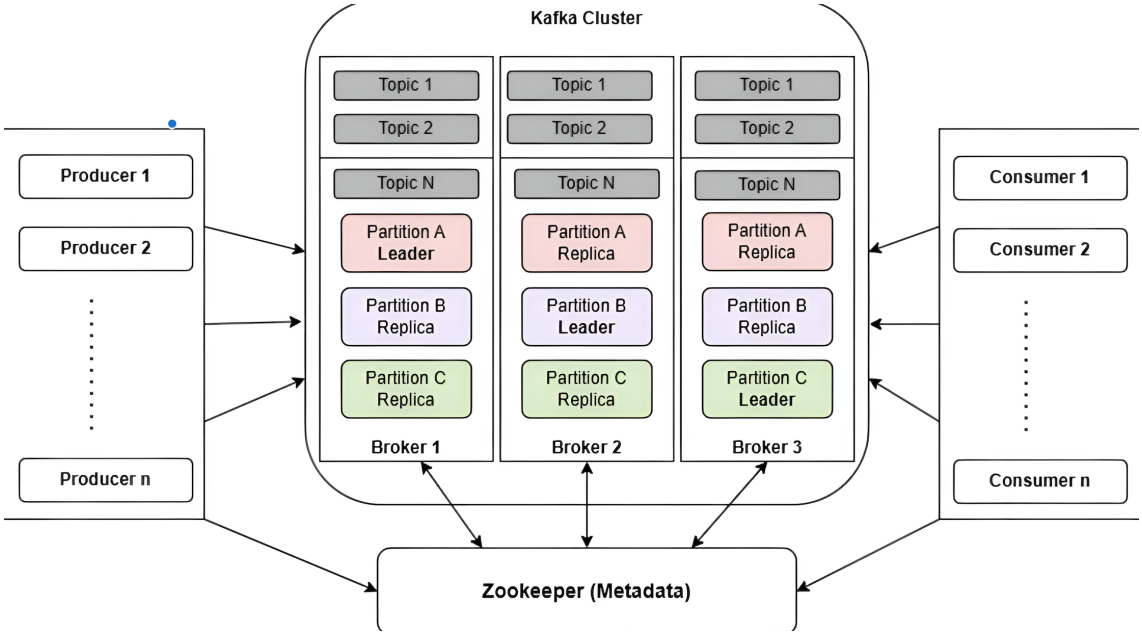

Figure 2.2: Kafka Architecture illustrating Producers, Consumers, Topics, Partitions, and Zookeeper [47].

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, making it particularly suited to complex, distributed Re-ID deployments.

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.

2.2.7 Containerization and orchestration

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.

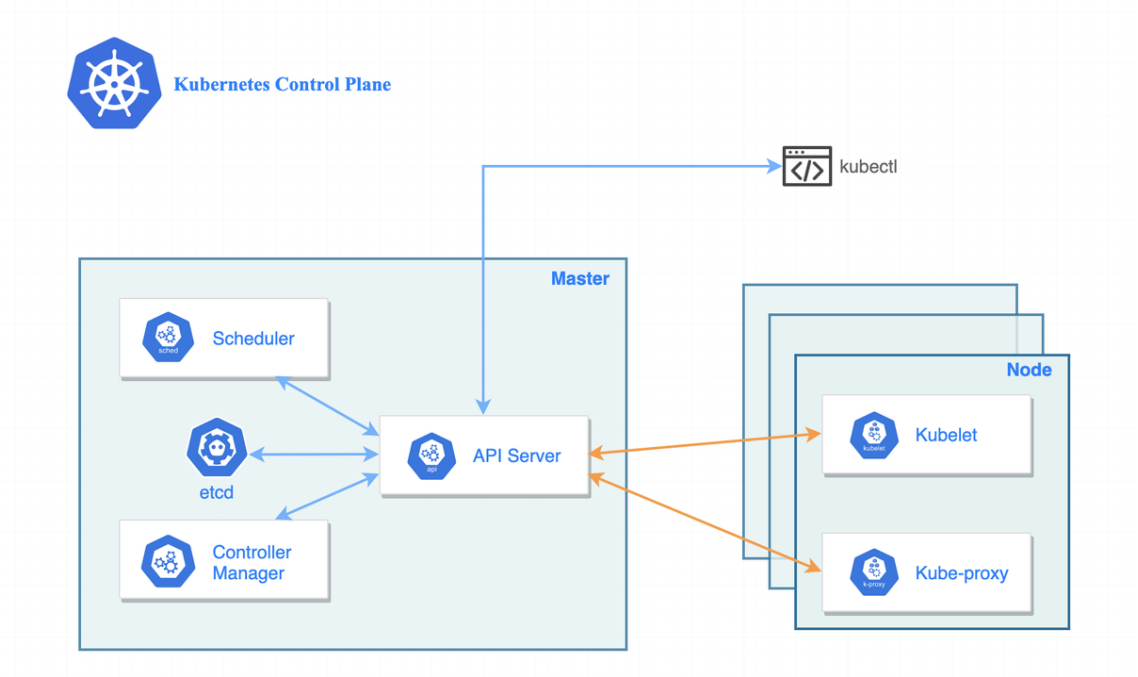

Figure 2.3: Kubernetes components [53].

Kubernetes complements Docker by orchestrating these containers at scale. It automates key operational tasks, including deployment, scaling, failover, and version rollouts across clusters. In the context of a hybrid edge-server person Re-ID pipeline, Docker is employed to package models, inference code, and APIs as distinct services. Kubernetes then orchestrates these services—such as Kafka brokers, Ray Serve instances, and vector databases—to ensure system-wide reliability, scalability, and streamlined version control.

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 [54].
- **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 [55].
- **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 [56].
- **Redis**, traditionally a key-value store, now includes vector search capabilities, offering speed and convenience; however, it provides limited support for advanced metadata filtering and complex query patterns [57].
- **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 [58].

Vector database indexes, such as HNSW [59], 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, **Qdrant** aligns most closely with the needs of our hybrid edge-server person Re-ID pipeline. Its capability to perform rapid cosine similarity searches with integrated metadata filtering—such as gender classification obtained via classification model — enables significant reductions in search space, thereby enhancing retrieval speed, accuracy, and overall computational efficiency.

CHAPTER 3. METHODOLOGY

Building upon the theoretical foundations established in Chapter 2, this chapter presents a comprehensive examination of (i) the person Re-ID module architecture and its integration within the hybrid edge-server management system, (ii) detailed specifications of the proposed lightweight Re-ID module optimized for SME deployment, (iii) edge device hardware implementation strategies, (iv) the deployment of microservices frameworks and (v) how the system utilizes person metadata (gender) to enhance the efficiency of identity retrieval processes.

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.

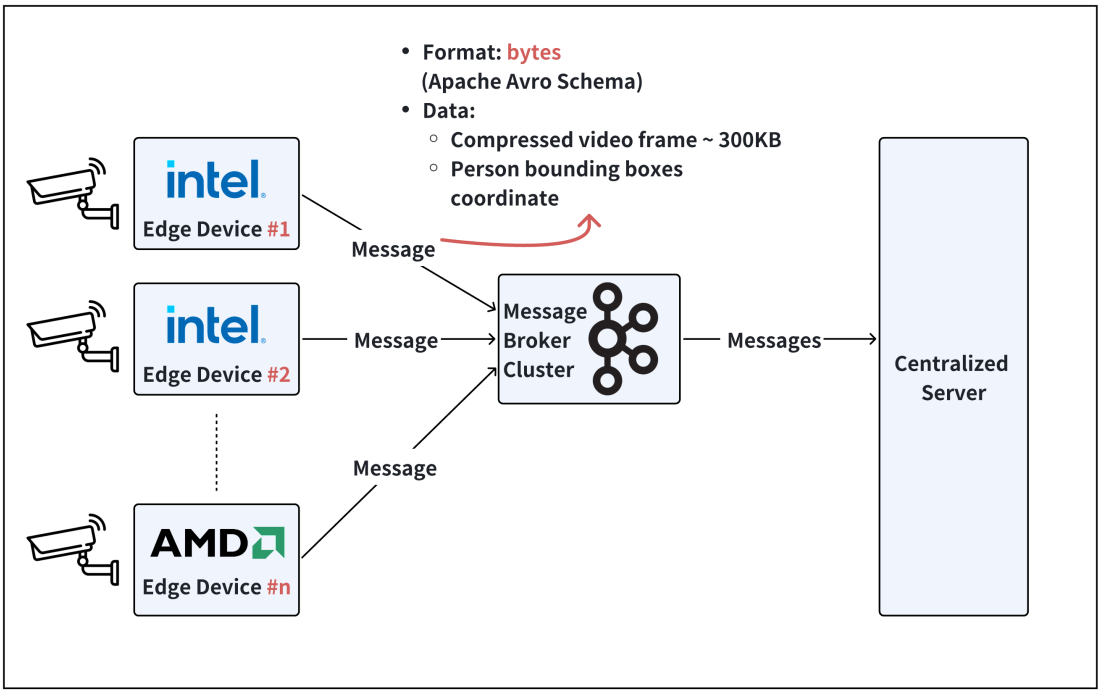

Figure 3.1: System overview of the hybrid edge-server person Re-ID pipeline showing the distributed setup with edge devices doing human detection, Kafka message broker cluster asynchronously handling communication, and central server managing model inference support for identity matching across multiple 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 x86 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 broker 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 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 14-20 FPS. Thus, with just 10 devices, the centralized server receives approximately 140-200 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 14 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 14 MB/s per device. Consequently, a system of 10 devices generates approximately 140 MB/s (1.12 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 3 brokers and 1 Zookeeper node is selected to ensure high availability and fault tolerance, as illustrated in Figure 3.2.

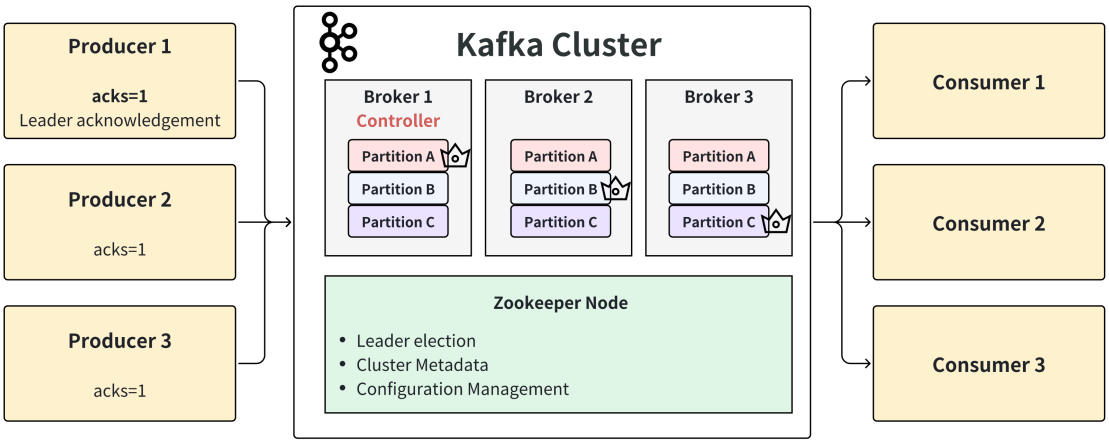

Figure 3.2: Kafka clusters with 3 brokers and 1 Zookeeper node.

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 Broker**: 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.
- **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.

3.2.2 Zookeeper

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, which is helpful for small to medium enterprises (SMEs) 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 Message serialization with Avro schema

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 follows this format:

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

Listing 3.1: 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"}
  ]
}
```

Listing 3.2: Avro schema definition for edge device messages

3.2.4 Topic settings

In this project, we setup 2 topics:

- `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).

```
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
```

Listing 3.3: Topic settings for `reid_input`

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
```

Listing 3.4: Topic settings for `reid_output`

The `reid_output` topic is configured with fewer partitions (3 vs 6) since output messages are typically aggregated results with lower frequency compared to continuous video frame input. The reduced retention time (6 hours vs 1 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.

3.2.5 Producer Settings

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,
)
```

Listing 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.

3.2.6 Consumer Settings

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.

```
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,
)
```

Listing 3.6: Kafka consumer configuration for 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 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_*)**: 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.

3.2.7 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

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

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)

3.3 Edge devices

The edge device processing pipeline presented in Figure 3.3 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.

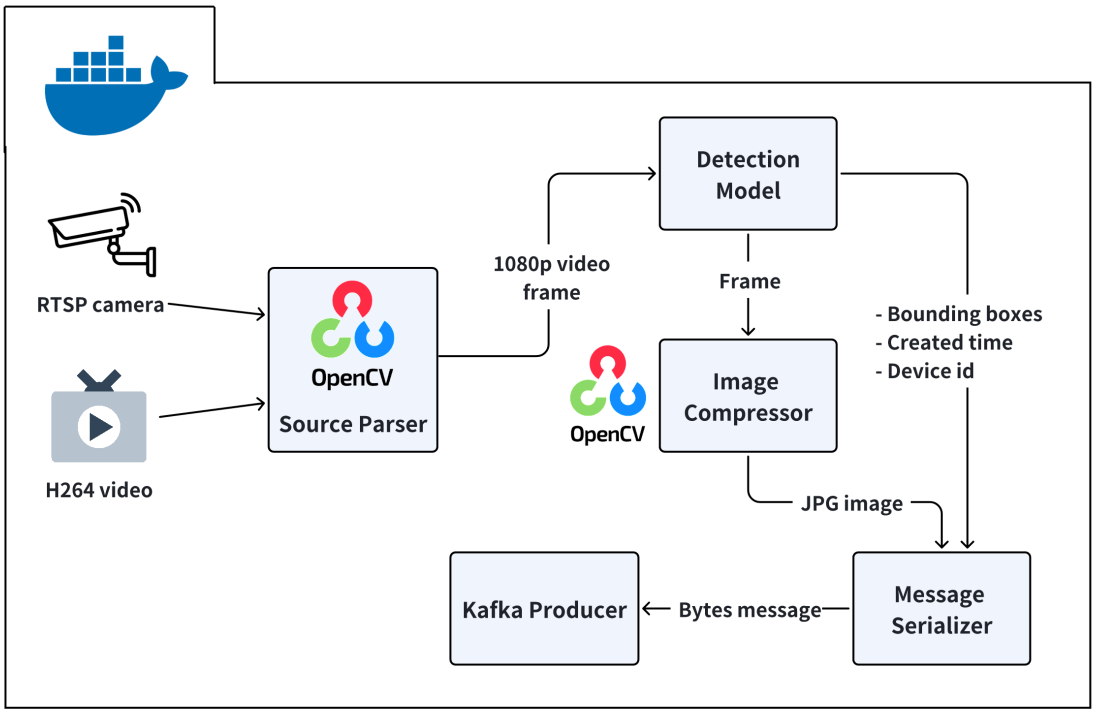

Figure 3.3: Edge devices processing pipeline

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

- **Source parser**: This component handles input from RTSP cameras delivering live video streams or, loaded from H264 video sources (demo video). 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.
- **Detection model**: The standardized video frames are then processed by a lightweight yet robust YOLOv11 object detection model, excellently suited for human detection. This detection 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.

The following sections provide detailed implementation descriptions for each component, beginning with the **Human detection with YOLOv11** module.

3.3.1 Human detection with YOLOv11

The YOLOv11 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 YOLOv11 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),
}
```

Listing 3.7: 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,
)
```

Listing 3.8: 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,
)
```

Listing 3.9: 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 **??**. 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 compression with OpenCV

As mentioned in Section 3.2.3, 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 to reduce the message size.

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])
```

Listing 3.10: 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 3.4: 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 3.2.3, the detection results along with the compressed images are structured into Avro schema serialized messages before being published 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()
```

Listing 3.11: 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:

```
# 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"]
```

Listing 3.12: 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 3.5. 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.

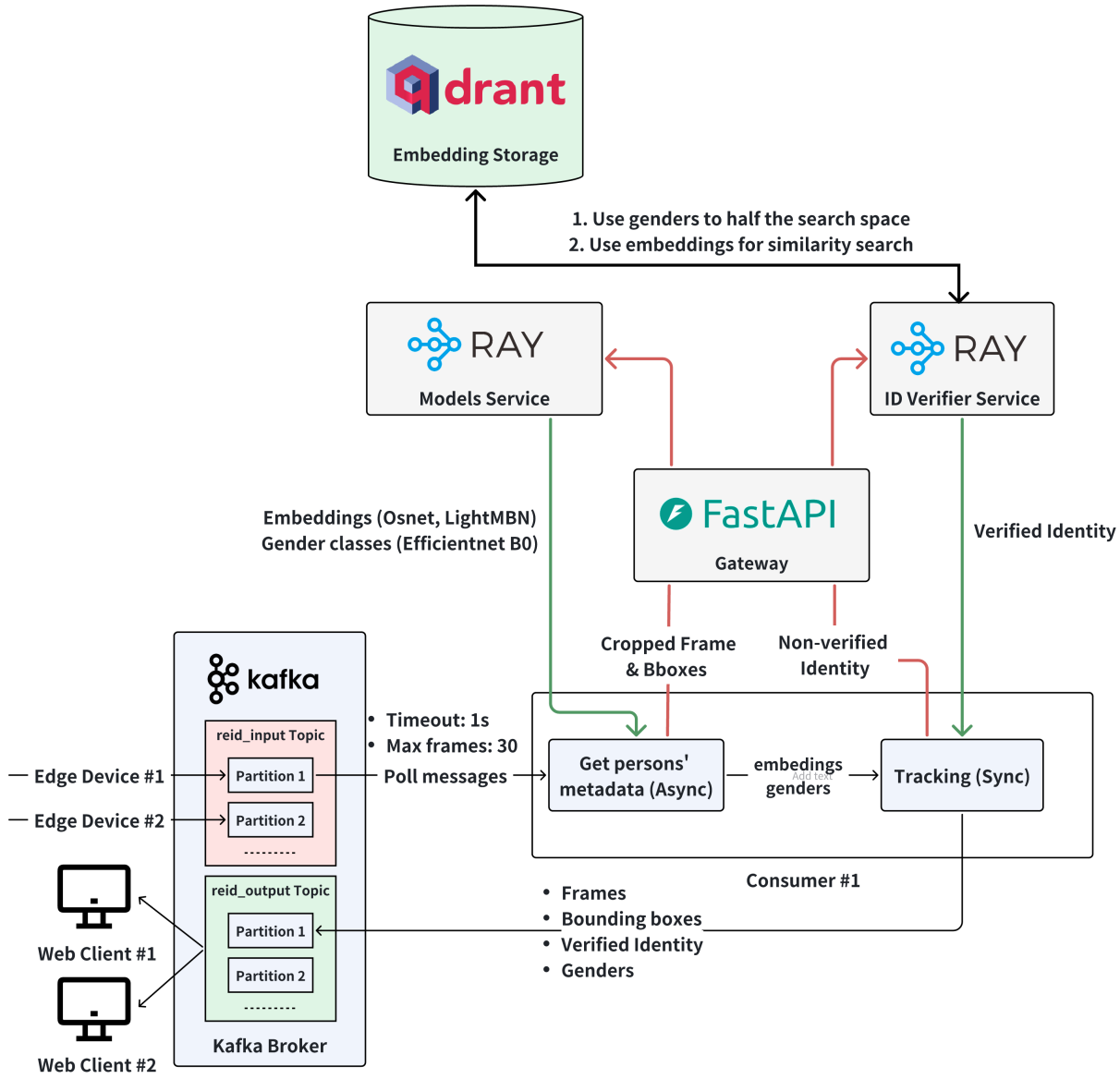

Figure 3.5: Centralized server architecture. The server is equipped with a decent GPU and CPU, and it is responsible for performing sophisticated model inference tasks, vector database operations, and multi-consumer message processing.

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 3.5, 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.

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—a critical requirement for the tracking

module of the Re-ID system. As illustrated in Figure 3.5, 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.

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

1. 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):
    - **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.
    - **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.
2. The processing goal is to obtain complete results for all frames in the batch before proceeding to tracking and identity matching operations.

The first processing step does not require maintaining frame order, as the primary objective is achieving accurate and efficient results that can be sorted by timestamp before performing tracking and identity matching. This independence from sequential processing makes asynchronous processing an optimal approach for the initial metadata extraction phase, allowing parallel computation of embeddings and gender classifications across multiple frames simultaneously to maximize throughput and minimize processing latency.

The **FastAPI Gateway** acts as the central coordination layer that orchestrates the processing pipeline between Kafka consumers and Ray Serve services. As depicted in Figure 3.5, when consumers poll batched messages (cropped frames and bounding boxes) from their assigned Kafka partitions, the processing follows a sequential workflow via the FastAPI Gateway. First, the **Get person's metadata (Async)** module sends all cropped frames in the batch to the **Models Service** for feature extraction (OSnet, LightMBN embeddings) and gender classification

(EfficientNet B0). After 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.

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 or BOTSort) 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 Qdrant 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.

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.

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:

```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
```

Listing 3.13: Ray Serve configuration file

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.

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, Feature extraction

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, Gender classification

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, as detailed in Section 4.1.

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. The training methodology, dataset composition, and performance evaluation are comprehensively documented in Section 4.1, where detailed experimental results demonstrate the model's effectiveness in distinguishing gender characteristics from person detection crops under various lighting conditions, viewing angles, and image qualities typical in surveillance environments.

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 Embedding Storage

a, Qdrant configuration

b, Retrieval approach

3.4.4 ID Verifier Service

3.4.5 Consumers

3.5 Web application

CHAPTER 4. EXPERIMENTAL RESULTS

4.1 Classification

4.2 Edge devices

CHAPTER 5. CONCLUSIONS AND FUTURE WORKS

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] Nishan Baral, *Kubernetes Architecture Explained in Layman's Terms*, 2023. **url**: https://www.linkedin.com/pulse/kubernetes-architecture-explained-laymans-terms-nishan-baral-ydjyc.

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

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

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

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

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

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