

Ph.D. DISSERTATION

Edge-Cloud Cooperative Platform
for Live Video Analytics
Applications

실시간 비디오 분석 응용을 위한
엣지-클라우드 협력적 플랫폼

FEBRUARY 2024

Department of Computer Science & Engineering
College of Engineering
Seoul National University

Juheon Yi

Edge-Cloud Cooperative Platform for Live Video Analytics Applications

Advisor Youngki Lee

Submitting a Ph.D. Dissertation of Engineering

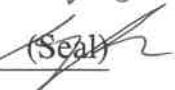
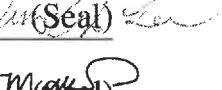
December 2023

Department of Computer Science & Engineering
College of Engineering
Seoul National University

Juheon Yi

Confirming the Ph.D. Dissertation written by Juheon Yi

February 2024

Chair:	Byung-Gon Chun	 (Seal)
Vice Chair:	Youngki Lee	 (Seal)
Examiner:	Jae W. Lee	 (Seal)
Examiner:	Sung-Ju Lee	 (Seal)
Examiner:	Matthai Philipose	 (Seal)

Abstract

Live video analytics enable various useful services including traffic monitoring, surveillance, person identification, and AR/MR. Despite the huge potential, enabling robust and efficient live video analytics is non-trivial. The core challenge lies in running the unique workload of analyzing the live video stream in real-time and seamlessly delivering the analysis results to the user for interaction on resource-constrained mobile devices. Such workload often requires a continuous and simultaneous execution of multiple Deep Neural Networks (DNNs) on high-resolution videos. In this dissertation, we (i) characterize the workload of emerging live video analytics apps, and (ii) design an edge-cloud cooperative platform to support the workload. Specifically, we perform end-to-end optimization across the edge, network, and cloud to support the workload with real-time throughput, low per-frame latency, and high accuracy.

We first design **EagleEye**, an AR system to identify missing person(s) in large, crowded urban spaces in real-time. Person identification imposes a unique workload of running a series of complex DNNs multiple times per each high-resolution video frame. Our key approach is *Content-Adaptive Parallel Execution*, which adapts the multi-DNN face identification pipeline depending on recognition difficulty (e.g., face resolution, pose) and cooperatively execute the workload at low latency using heterogeneous processors on mobile and cloud. We also design a novel ICN and its training methodology that utilize the probes of the target to recover missing facial details in the LR faces to improve the accuracy of the state-of-the-art face identification techniques. Our results show that ICN significantly enhances LR face recognition accuracy (true positive by 78% with only 14% false positive), and **EagleEye** accelerates the latency by 9.07 \times with only 108 KBytes of data offloaded to the cloud.

We next design **Pendulum**, an end-to-end live video analytics system with a novel network-compute joint scheduling. In practical scenarios, resource bottleneck frequently alternates across the network (video streaming) and the compute (DNN inference) stages due to dynamic scene content and resource availability. However, prior single-stage scheduling systems suffer from throughput/accuracy fluctuation and resource wastage due to over-provisioning. To overcome the limitations, we newly discover the interplay between video bitrate and DNN complexity. Based on this, we design an end-to-end system composed of (i) a joint scheduling mechanism (to estimate network, compute resource demands and availability as well as control resource usages) and (ii) a joint resource scheduler. Evaluation with various videos and state-of-the-art DNNs show that **Pendulum** achieves up to 0.64 mIoU gain and 1.29 \times higher throughput than state-of-the-art baselines. **Pendulum** also achieves near-optimal multi-user resource scheduling performance with minimal search overhead, achieving 25% cost reduction compared to network-compute decoupled scheduling baseline.

Finally, we design **Heimdall**, a mobile platform to support multi-DNN and rendering concurrency on mobile GPUs. We analyze that existing mobile deep learning frameworks are designed for single DNN execution in isolated environments and fail to support multi-DNN and rendering concurrent workload (e.g., inference latency increases from 59.93 to 1181 ms, rendering frame rate drops from 30 to 12 fps). While multi-task scheduling has been actively studied for desktop GPUs (e.g., parallelization, preemption), applying it to mobile GPUs is challenging due to limited architectural support and memory bandwidth. To tackle the challenge, we design a *Pseudo-Preemption* mechanism which i) breaks down the bulky DNN into smaller units, and ii) prioritizes and flexibly schedules concurrent GPU tasks. **Heimdall** efficiently supports multiple MR app scenarios, enhancing the frame rate from 11.99 to 29.96 fps while reducing the

worst-case DNN inference latency by up to $\approx 15\times$ compared to the baseline multi-threading approach.

keywords: Live video analytics, Edge-cloud cooperative system, Mobile/edge AI

student number: 2020-39481

Contents

Abstract	i
Contents	iv
List of Tables	x
List of Figures	xi
1 Introduction	1
1.1 Motivation and Challenges	1
1.2 Proposed Edge-Cloud Cooperative Platform	3
1.2.1 Design Goals	3
1.2.2 Platform Architecture	4
1.2.3 Key Solutions	5
1.3 Contributions	6
1.4 Dissertation Overview	7
2 Motivational Studies	8
2.1 Applications and Requirements	8
2.1.1 Application Scenarios	8
2.1.2 Workload Characterization	9
2.2 Challenges	10
2.2.1 Complexity of the State-of-the-art DNNs	10

2.2.2	Large Data Size and Compute of Each Analysis Task	12
2.2.3	Alternating Resource Bottleneck from Dynamic Resource Availability and Workload	13
2.2.4	Multi-Task Resource Contention	15
3	Related Work	18
3.1	Live Video Analytics Applications	18
3.2	On-Device Systems	18
3.2.1	Mobile Deep Learning Frameworks	18
3.2.2	On-Device Continuous Mobile Vision	19
3.3	Cloud Offloading Systems	19
3.3.1	Offloading for Continuous Mobile Vision	19
3.3.2	Adaptive Bitrate for Live Video Analytics	20
3.3.3	ML Serving in Edge/Cloud Server	20
3.3.4	Edge-Cloud cooperative Inference Systems	20
3.4	Tiny ML/Efficient Deep Learning	21
4	EagleEye: AR-based Person Identification in Crowded Urban Spaces	22
4.1	Introduction	22
4.2	Motivating Scenarios	27
4.3	Preliminary Studies	28
4.3.1	How Fast Can Humans Identify Faces?	28
4.3.2	How Accurate Can DNNs Identify Faces?	30
4.3.3	How Fast Can DNNs Identify Faces?	32
4.3.4	Summary	33
4.4	EagleEye: System Overview	33
4.4.1	Design Considerations	33
4.4.2	Operational Flow	34

4.5	Identity Clarification-Enabled Face Identification Pipeline	35
4.5.1	Face Detection	36
4.5.2	Identity Clarification Network	36
4.5.3	Face Recognition and Service Provision	40
4.6	Real-Time Multi-DNN Execution	41
4.6.1	Workload Characterization	41
4.6.2	Content-Adaptive Parallel Execution	42
4.7	Implementation	47
4.8	Evaluation	48
4.8.1	Experiment Setup	48
4.8.2	Performance Overview	50
4.8.3	Identity Clarification Network	51
4.8.4	Content-Adaptive Parallel Execution	53
4.8.5	Performance for Varying Crowdedness	55
4.8.6	Performance on Other Mobile Devices	56

5	Pendulum: Network-Compute Joint Scheduling for Efficient and Accurate Live Video Analytics	57
5.1	Introduction	57
5.2	Limitations of Prior Works	61
5.2.1	Limitations of Single-Stage Scheduling	61
5.2.2	Why Simple Combination of Two Schedulers Fails?	62
5.3	Our Approach	63
5.3.1	Goals	63
5.3.2	Key Idea: Joint Scheduling	64
5.3.3	Why is Joint Scheduling Possible?	65
5.3.4	Joint Scheduling Problem Formulation	67
5.4	Design Overview	68
5.4.1	Challenges	68

5.4.2	Key Ideas	69
5.4.3	System Architecture	69
5.5	Joint Scheduling Mechanism	70
5.5.1	Network-Compute Demand Profiler	71
5.5.2	Resource Availability Monitor	74
5.5.3	Joint Scheduling Knob Controller	76
5.6	Joint Scheduling Algorithm	77
5.7	Evaluation	80
5.7.1	End-to-End Improvement	82
5.7.2	Joint Scheduling on SOTA Systems	84
5.7.3	Performance on Various App Settings	84
5.7.4	Performance in Compute Bottleneck	85
5.7.5	Microbenchmarks	86
6	Heimdall: Mobile GPU Coordination Platform for AR Applications	89
6.1	Introduction	89
6.2	Analysis on GPU Contention	93
6.3	Heimdall System Overview	95
6.3.1	Approach	95
6.3.2	Design Considerations	97
6.3.3	System Architecture	98
6.4	Preemption-Enabling DNN Analyzer	99
6.4.1	Overview	99
6.4.2	Latency Profiling	100
6.4.3	DNN Partitioning	101
6.5	Pseudo-Preemptive GPU Coordinator	102
6.5.1	Overview	102
6.5.2	Utility Function	103

6.5.3	Scheduling Problem and Policy	104
6.5.4	Greedy Scheduling Algorithm	106
6.6	Additional Optimizations	107
6.6.1	Preprocessing and postprocessing	108
6.6.2	CPU Fallback Operators	108
6.7	Implementation	109
6.8	Evaluation	110
6.8.1	Experiment Setup	110
6.8.2	Performance Overview	110
6.8.3	DNN Partitioning/Coordination Overhead	112
6.8.4	Pseudo-Preemptive GPU Coordinator	112
6.8.5	Performance for Various App Scenarios	114
6.8.6	DNN Accuracy	114
6.8.7	Energy Consumption Overhead	116
7	Conclusion	117
7.1	Summary	117
7.2	Discussion	118
7.2.1	Scalability and Generality of EagleEye to Other Workloads	118
7.2.2	Generality of Pendulum to Wider Network and System Environments	119
7.2.3	Impact of Hardware Evolution on Heimdall	120
7.3	Future Projection: Would the Importance of System Optimization Persist?	122
7.3.1	DNN complexity increase vs. hardware evolution	122
7.3.2	Video bitrate increase vs. network evolution	123
7.4	Future Works	125
7.4.1	System Support for 3D Point Cloud Videos	125
7.4.2	App-RAN Cross-Layer Optimization	125

List of Tables

1.1	Challenges and our proposed solutions	6
2.1	DNN and rendering requirements for the example MR app scenarios.	10
2.2	DNNs for the above MR apps. Inference time is measured on MACE over LG V50 (Adreno 640 GPU).	11
2.3	Complexity comparison between state-of-the-art DNNs and backbones.	12
4.1	Inference time of DNNs with TensorFlow-Lite running on LG V50 (Qualcomm Adreno 640 GPU).	32
4.2	Complexity and latency of component DNNs. FLOPs are measured with <i>tf.profiler.profile()</i> function.	32
4.3	Average and standard deviation of the composition of each face type in the test dataset.	48
5.1	Evaluation datasets.	81
6.1	Face detection and person segmentation accuracy (IoU) for the AR emoji scenario.	115

List of Figures

1.1	Live video analytics application scenarios.	2
1.2	Edge-cloud cooperative platform architecture.	5
2.1	Offloading latency of multi-DNN face identification pipeline.	12
2.2	Example bottleneck (colored in red) timelines.	14
2.3	Bitrate and workload (# of objects) for different scenes.	14
2.4	Multi-DNN GPU contention.	16
2.5	Rendering-DNN GPU contention on MACE over LG V50 (immersive online shopping scenario).	17
2.6	Rendering-DNN GPU contention on TF-Lite over Google Pixel 3 XL (criminal chasing scenario).	17
4.1	Example usage scenario of EagleEye: parent finding a missing child. More examples in Chapter 4.2.	23
4.2	Multi-DNN face identification pipeline.	24
4.3	Human cognitive abilities on identifying faces in crowded scenes: response time and accuracy.	29
4.4	Face verification accuracy.	31
4.5	Latency of face identification pipeline.	31
4.6	Feature map visualization for varying resolutions (points with same color represents same identity).	31

4.7	Operation of EagleEye in a nutshell.	34
4.8	EagleEye system overview.	35
4.9	GANs reconstruct realistic faces, but fail to preserve the face identity.	36
4.10	Identity Clarification Network: overview.	37
4.11	Generator network architecture.	37
4.12	CDF of face distances for varying resolutions.	39
4.13	Edge-based background filtering.	43
4.14	Variation-Adaptive Face Recognition.	44
4.15	Spatial Pipelining on heterogeneous processors.	46
4.16	In-the-wild dataset examples.	48
4.17	EagleEye performance overview.	50
4.18	Performance of Identity Clarification Network.	51
4.19	Feature map visualization for ICN.	51
4.20	Reconstruction example of ICN.	52
4.21	Background filtering.	52
4.22	Example operation of Edge-Based Background Filtering.	53
4.23	Performance of Variation-Adaptive Face Recognition.	54
4.24	Spatial Pipelining performance.	54
4.25	End-to-end latency for varying crowdedness.	55
4.26	Latency evaluation on Google Pixel 3 XL.	56
5.1	Scenario: cloudlet-based person monitoring.	58
5.2	Performance analysis of network-only scheduling with EAAR [1] (b: bottleneck, nb: no bottleneck).	61
5.3	Accuracy changes while running decoupled schedulers. C-1/2 and N-1 denote compute and network scheduling events, respectively.	62
5.4	Single-stage vs. joint scheduling comparison.	64
5.5	Joint scheduling example for network bottleneck scenario.	65

5.6	Illustration of the impact of the receptive field.	65
5.7	Example detection results (box and confidence) for different crop sizes around ground truth (GT) box.	65
5.8	Detection accuracy in low-bitrate video.	66
5.9	Segmentation accuracy in low-bitrate video.	67
5.10	Pendulum system architecture (Yellow: video analytics pipeline, Green: Pendulum components).	70
5.11	Demand curves for different scenes. Blue/red points: configs above/below the accuracy requirement, green curve: Pareto-optimal configs. .	71
5.12	Accuracy gain from DNN backbone increase (D0 to D6) varies depending on video bitrate.	73
5.13	App-side packet-level bandwidth estimation cannot know RAN's remaining bandwidth capacity.	75
5.14	RIC message format.	76
5.15	Iterative Max Cost Gradient algorithm example (2 iterations, CG : cost gradient).	76
5.16	Bandwidth required to compensate Δ_t inference latency differs depending on the demand curve.	78
5.17	Throughput-accuracy comparison in network bottleneck scenario. .	82
5.18	Testbed implementation.	82
5.19	Over-the-air performance.	82
5.20	Frame-wise latency comparison.	82
5.21	Joint scheduling on state-of-the-art systems.	83
5.22	Performance across various tasks & DNNs.	84
5.23	Performance for (res, backbone) knobs.	85
5.24	Performance in compute bottleneck.	85
5.25	Operation timeline when the compute becomes a bottleneck. .	86
5.26	Demand profiler performance breakdown.	86

5.27	Performance under bandwidth fluctuation.	86
5.28	Impact of profiling interval on accuracy.	86
5.29	Performance of accuracy modeling.	87
5.30	Multi-user scheduling performance.	87
6.1	Multi-DNN GPU contention example.	94
6.2	System Architecture of Heimdall .	98
6.3	DNN inference latency with and without camera.	100
6.4	Operator-level latency distribution.	100
6.5	Camera frame rendering latency.	100
6.6	Example DNN latency profiling result on Google Pixel 3 XL.	100
6.7	Operation of DNN partitioning.	101
6.8	DNN inference latencies for varying partition sizes.	101
6.9	End-to-end DNN inference pipeline example for RetinaFace [2] detector.	108
6.10	Performance overview of Heimdall on LG V50.	111
6.11	DNN partitioning overhead.	112
6.12	Performance comparison of GPU coordination policies.	113
6.13	Opportunistic CPU offloading performance.	114
6.14	Performance of Heimdall for other AR app scenarios.	115
7.1	DNN inference complexity (assuming 1080p@30fps input) vs. hardware capability (NPU: Apple A12-A17, GPU: Qualcomm Adreno 630-740).	122
7.2	Video bitrate vs. network bandwidth (Green: target per-user experienced bandwidths of 4G, 5G, 6G, Blue: actual measurements from existing traces [3, 4] and our own measurements).	124

Chapter 1

Introduction

1.1 Motivation and Challenges

Live video analytics is an emerging class of applications (apps) which analyzes the live video streams from mobile devices (e.g., smartphones, AR glasses, CCTVs), delivers the analyzed results to the users, and enables real-time user interaction. It enables various services including traffic monitoring [5], surveillance [6], person identification [7], and Augmented Reality (AR)/ Mixed Reality (MR) [1,8] (see Chapter 2.1.1 for detailed scenarios).

Despite the potential, realizing robust and efficient live video analytics apps is highly challenging. The core challenge lies in supporting the unique workload of analyzing the live video stream in real-time and seamlessly delivering the analysis results to the user for interaction on resource-constrained mobile devices. Specifically, live video analytics app has the following computational requirements. First, it needs to accurately analyze the live video stream as well as the user behaviors (e.g., hand, gaze movement) for interaction, which often requires a continuous and simultaneous execution of multiple Deep Neural Networks (DNNs) on high-resolution video streams (see Table 2.1). Second, it should seamlessly synthesize and render the analysis results (e.g., bounding

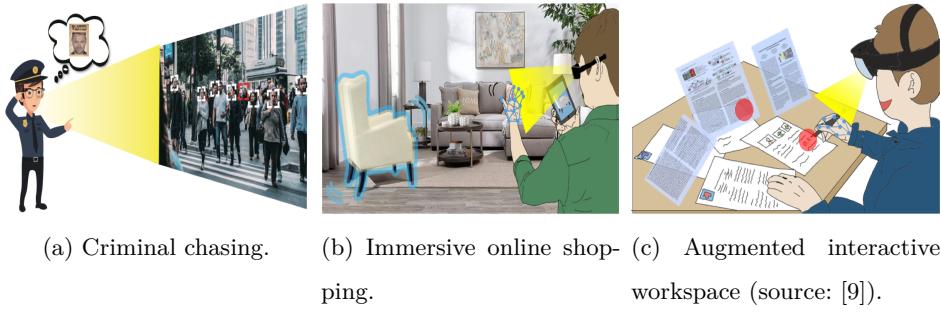


Figure 1.1: Live video analytics application scenarios.

boxes and trajectories over the video frames, virtual objects) over the analyzed scenes for immersive user experiences. Finally, background DNN inference computation and foreground UI rendering should be simultaneously performed in real-time under resource constraints.

Supporting such concurrent multi-DNN and rendering workload incurs the following technical challenges.

Challenge #1: Large data size and compute requirements per each task (Chapter 2.2.2). Each analysis task requires a repetitive execution of multiple DNNs over high-resolution videos. Due to large input data size and high compute complexity of multiple DNNs, it is non-trivial to run the workload both through on-device and cloud offloading. For example, identifying distant faces in criminal chasing scenario requires face detection on 1080p frame, and face recognition per each face. It takes 8.6s and 3.4s for a frame with 17 faces, mainly due to large number of DNN inferences and high network transmission latency due to large frame size.

Challenge #2: Alternating bottlenecks from dynamic resource and workload fluctuation (Chapter 2.2.3). The workload and resource availability independently fluctuate both within and across the edge device, network, and cloud server. This incurs a complex, alternating resource bottleneck patterns, incurring severe latency/accuracy fluctuation. For example, assume run-

ning a multi-DNN person analysis workload (composed of a person detector and two classifiers for face and action, see Chapter 2.2.3 for details) [10] through cloud offloading. Compute workload fluctuates depending on scene complexity (i.e., number of objects in the scene). Network bandwidth fluctuates due to user mobility and wireless channel status [11–13]. Available GPU utilization also fluctuates (independently of bandwidth) due to multi-user resource contention, causing inference latency slowdown [14]. The two factors fluctuate independently of each other, causing alternating resource bottleneck patterns; for example, network and compute bottlenecks occur 3 and 2 times each in a non-overlapping and switching pattern within a 30 seconds window.

Challenge #3: Multi-task resource contention (Chapter 2.2.4). Running multiple DNNs and rendering tasks concurrently incurs resource contention and degrades performance. This is especially severe on resource-constrained mobile devices (especially mobile GPUs). For example, running 4 DNN tasks (object detection, segmentation, hand tracking, and image style transfer) and 1080p@30 fps video frame rendering tasks concurrently in Mixed Reality shopping scenario increases the inference latency from 59.93 ± 3.68 to 1181 ± 668 , and drops the rendering frame rate to as low as 11.99 fps.

1.2 Proposed Edge-Cloud Cooperative Platform

1.2.1 Design Goals

Specifically, our platform aims to achieve the following design goals:

- **High throughput and low latency.** We aim at end-to-end scheduling across the end users and analysis server for real-time video processing (e.g., 30 fps throughput) while satisfying the app-specified accuracy requirements. We also aim at soft real-time latency (e.g., <100 ms) so that the analysis

result is delivered to users promptly for further actions (e.g., bounding box displayed on screen for officers to confirm with his own eyes).

- **Robustness under dynamic resource and workload.** We also aim at robust throughput, latency, and accuracy performance under dynamic resource availability and workload fluctuation. We aim at achieving the goal with end-to-end optimization across the edge device, network, and edge/cloud server, as well as cross-layer optimization across the application, framework, and the OS stack.
- **High scalability and generality.** Finally, our goal is to develop a platform that is highly scalable and generalizable across various input video sources, analysis tasks, and number of users.

1.2.2 Platform Architecture

Figure 1.2 shows the architecture of our edge-cloud cooperative platform. Given the input source video stream (e.g., camera, LiDAR) and the app specification (video analytics pipeline, app Service Level Objectives (SLOs), and DNN models for each analysis task), the platform cooperatively utilizes the mobile/edge device and the edge/cloud server to run the workload. We currently target edge devices embedded with accelerators capable of running the DNN inferences (e.g., Google Pixel 4 smartphone with Qualcomm Adreno 640 GPU and Google Tensor TPU, Magic Leap One AR glasses with NVIDIA Jetson TX2 Pascal GPU, or Oculus Quest VR headset with Adreno 540 GPU). However, our platform can also be deployed on devices without on-device accelerators (e.g., CCTVs, microcontrollers) for cloud offloading-based video analytics.

Specific operation of the platform is as follows. First, *Content-Aware Load Distributor* (i) analyzes the input content and determines the analysis DNNs based on its difficulty, and (ii) distributes the workload across the edge device

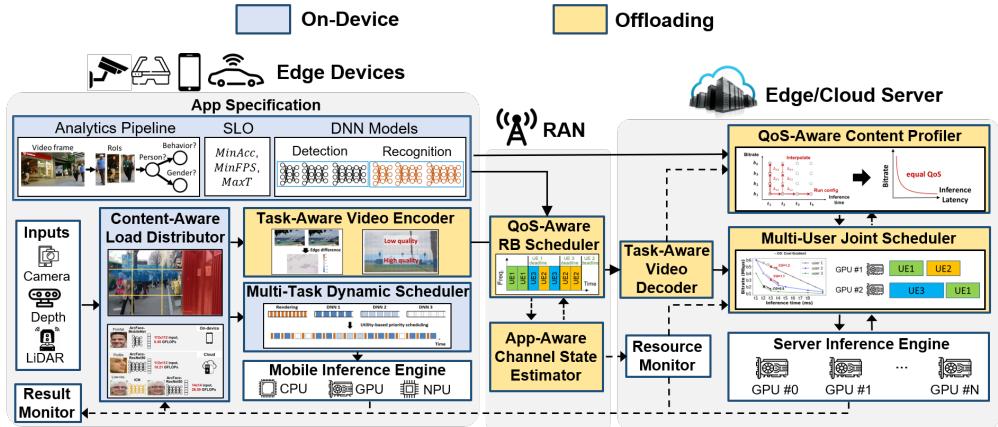


Figure 1.2: Edge-cloud cooperative platform architecture.

and cloud depending on resource availability. For the workload scheduled for offloading, the *Task-Aware Video Encoder* efficiently compresses the video with minimal task accuracy drop. The network (video streaming) and compute (DNN inference) stages of the offloading pipeline is scheduled jointly by the *QoS-Aware RB Scheduler* and the *Multi-User Joint Scheduler*, which jointly controls the video encoding bitrate and the inference DNN complexity depending on the network/compute resource availability and input scene content profiled by the *QoS-Aware Content Profiler*. Finally, the *Multi-Task Dynamic Scheduler* utilizes multiple mobile processors (e.g., CPU, GPU, NPU) to schedule the executions of multi-DNN and rendering tasks distributed for on-device inference.

1.2.3 Key Solutions

Table 1.1 summarizes our platform’s key features and solutions to tackle the aforementioned challenges.

- **Content-aware adaptation and cooperative execution.** We design EagleEye, which incorporates a novel content-aware adaptation and cooperative inference approach to execute the multi-DNN face identification pipeline over

Table 1.1: Challenges and our proposed solutions.

Challenges	Solutions
#1: Large data size and compute	#1: Content-aware adaptation and cooperative execution (Chapter 4 - EagleEye)
#2: Alternating resource bottlenecks	#2: Network-compute joint scheduling (Chapter 5 - Pendulum)
#3: Multi-task resource contention	#3: App-aware concurrency support on mobile GPUs (Chapter 6 - Heimdall)

high-resolution video in low latency.

- **Network-compute joint scheduling.** We design Pendulum, which identifies a novel tradeoff relationship between network (video streaming) and compute (DNN inference) stages in the cloud offloading pipeline and leverages the joint scheduling opportunity to guarantee robust performance under alternating resource bottlenecks.
- **App-aware multi-DNN/rendering concurrency support.** We design Heimdall, which utilizes app-aware (e.g., DNN model architecture and latency SLO) task partitioning and supports fine-grained scheduling of concurrent multi-DNN and rendering tasks to minimize resource contention achieve high latency SLO satisfaction rate on resource-limited mobile devices, especially on mobile GPUs.

1.3 Contributions

This dissertation thoroughly characterizes the concurrent multi-DNN and rendering workload of emerging live video analytics apps (e.g., Mixed Reality) and their requirements. Based on it, we design an edge-cloud cooperative platform to support the workload. While there have been several systems for live video analytics (e.g., on-device AI systems, adaptive bitrate streaming systems, and

cloud ML inference serving systems as analyzed in Chapter 3), they have been mostly limited to single-stage optimization. Our work is clearly distinguished in that we perform a comprehensive, end-to-end optimization across the edge, network, and cloud for high throughput, low latency, and high accuracy. We implement our platform using commercial devices (smartphones and edge servers), and conduct extensive real-world evaluation to validate their effectiveness (e.g., up to $9.07\times$ per-frame latency reduction, 30 fps throughput with 0.64 higher accuracy (mIoU), and up to $15\times$ worst-case inference latency reduction).

1.4 Dissertation Overview

The rest of the dissertation is organized as follows. We characterize the workloads of future live video analytics applications, and analyze challenges in supporting the workload in Chapter 2. We then summarize prior works and their limitations in Chapter 3. Chapters 4–6 details our systems to realize our proposed edge-cloud cooperative platform. Finally, Chapter 7 summarizes discussion and future works.

Chapter 2

Motivational Studies

Our dissertation mainly focuses on MR as representative apps. We first conduct motivational studies to characterize the workloads of futuristic MR apps (Chapter 2.1.1), and analyze the core challenges in supporting the workload (Chapter 2.2).

2.1 Applications and Requirements

2.1.1 Application Scenarios

Criminal chasing (Figure 1.1(a)). A police officer chasing a criminal in a crowded space (e.g., shopping mall) sweeps the mobile device to take a video of the area from distance. The mobile device processes the video stream to detect faces and find the matching one with the criminal. Specifically, it continuously runs face detection per scene and face recognition per each detected face. Detection results are seamlessly overlayed on top of the camera frames and rendered on screen to narrow down a specific area to search.

Immersive online shopping (Figure 1.1(b)). An online shopper wearing MR glasses positions a virtual couch in his room to see if the couch matches well before buying it. The MR glasses analyze the room by detecting its layout

and furniture, and renders the couch in a suitable position. The user can also change the style of the couch (e.g., color, texture), as well as adjust the arrangement with his hand movements. This app requires i) running object detection and image segmentation simultaneously to analyze the room, ii) running hand tracking and image style transfer to recognize user's hand movements and adjust the style of the couch, and iii) rendering the virtual couch on the right spot seamlessly.

Augmented interactive workspace (Figure 1.1(c)). A student wearing MR glasses creates an interactive workspace by combining the physical textbooks and virtual documents. When he encounters a concept he does not understand, he commands the MR glasses to search for related documents on the web via hand gestures. The searched documents are augmented near the textbooks. Also, the note he makes on the textbooks is recognized and saved as a digital file in his device for future edits. This app runs hand tracking and text detection, while seamlessly rendering the virtual documents.

Other multi-DNN MR apps include MR emoji [15] (face detection + segmentation + style transfer) or surroundings monitoring for visual support [16] (object and face detection + pose estimation).

2.1.2 Workload Characterization

Real-time, concurrent multi-DNN execution. The core of MR apps is accurately analyzing the physical world and user behavior to combine the virtual contents, which requires running multiple DNNs concurrently (see Tables 2.1 and 2.2 for examples). Also, such analysis needs to be continuously performed over a stream of images to seamlessly generate and overlay the virtual contents, especially in fast-changing scenes (e.g., criminal chasing). Moreover, DNNs need to run over high-resolution inputs for accurate analysis (e.g., recognizing small hand-writings or distant faces requires over 720p or 1080p frames [7,17]). These

Table 2.1: DNN and rendering requirements for the example MR app scenarios.

	Criminal chasing	Immersive online shopping	Augmented interactive workspace	MR emoji
Continuously executed DNNs (fps)	<ul style="list-style-type: none"> - Face detection [2] - Face recognition [21] (< 1s per scene) 	<ul style="list-style-type: none"> - Image segmentation [22] (1-5 fps) - Object detection [23] (1-5 fps) - Hand tracking [24] (1-10 fps) 	<ul style="list-style-type: none"> - Text detection [17] (1-5 fps) - Hand tracking [24] (1-10 fps) 	<ul style="list-style-type: none"> - Face detection [2] (1-10 fps) - Image segmentation [22] (1-10 fps)
Event-driven DNNs (response time)		<ul style="list-style-type: none"> - Image style transfer [25] (< 0.1s) 		<ul style="list-style-type: none"> - Image style transfer [25] (< 0.1s)
Rendering (resolution, fps)	<ul style="list-style-type: none"> - Camera frames (1080p, 30 fps)¹ - Bounding boxes 	<ul style="list-style-type: none"> - Couch (1440p, 60 fps)² 	<ul style="list-style-type: none"> - Virtual documents (1440p, 60 fps)² - Handwriting updates 	<ul style="list-style-type: none"> - Camera frames (1080p, 30 fps)¹ - Emoji/character mask

^{1,2} Microsoft HoloLens 2 [26] can record 1080p videos at 30 fps, and display 1440p resolution at 60 Hz at maximum.

characteristics are clearly distinguished from prior works [16, 18–20] that have mostly considered running a single DNN over simple scenes with a few main objects that can be analyzed with smaller resolution (e.g., 300×300).

Seamless rendering on top of concurrent DNN execution. MR apps need to seamlessly augment the virtual contents over the analyzed scenes for immersive user experiences. Such foreground rendering should be continuously performed in real-time in presence of the multi-DNN execution, causing serious contention on resource-constrained mobile GPUs.

Summary. Concurrent execution of multi-DNN and rendering necessitates a platform to prioritize and coordinate their execution on the mobile GPU. Careful coordination will become more important if an app requires audio tasks (e.g., voice command recognition, spatial audio generation) along with the vision tasks, or higher frame rate for more immersive user experience.

2.2 Challenges

2.2.1 Complexity of the State-of-the-art DNNs

One might think that multi-DNN execution on mobile devices is becoming less challenging due to the emergence of lightweight model architectures (e.g., Mo-

Table 2.2: DNNs for the above MR apps. Inference time is measured on MACE over LG V50 (Adreno 640 GPU).

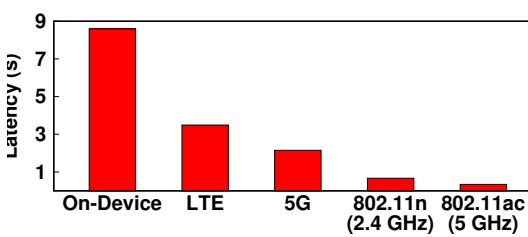
Task	Model	Input size	CPU/GPU ops	Inference time
Object detection	YOLO-v2 [23]	416×416×3	0/33	95 ms
Face detection	RetinaFace [2]	1,920×1,080×3	6/129	230 ms
Face recognition	ArcFace [21]	112×112×3	0/106	149 ms
Image segmentation	DeepLab-v3 [22]	513×513×3	0/101	207 ms
Image style transfer	StyleTransfer [25]	640×480×3	14/106	60 ms
Pose estimation	CPM [27]	192×192×3	0/187	14 ms
Hand tracking	PoseNet [24]	192×192×3	0/74	256 ms
Text detection	EAST [17]	384×384×3	8/117	214 ms

bileNet [28,30]) and the increasing computing power of mobile GPUs. However, the challenge still exists. The main reason is that state-of-the-art DNNs do not employ the lightweight models directly, but enhance them with complex task-specific architectures to achieve higher accuracy.

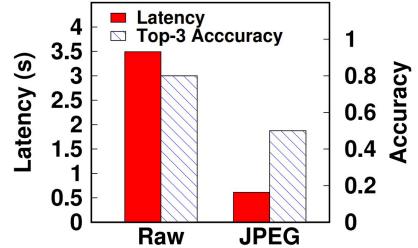
Table 2.3 compares the complexity of state-of-the-art DNNs with their backbones in terms of floating-point operations (FLOPs) required for a single inference. The reported values are either from the original paper if available, or profiled with `TensorFlow.Profiler.Profile()` function. Overall, state-of-the-art DNNs require 5.76-10.75× FLOPs than their backbones, showing that the lightweight backbone is only a small part of the whole model. For instance, RetinaFace [2] detector employs feature pyramid [31] on top of MobileNet-v1 [28] to accurately detect tiny faces, whereas ArcFace [21] recognizer adds batch normalization layers on ResNet [29] and replaces 1×1 kernel to 3×3 for higher accuracy. Similar holds for DeepLab-v3 [22] (segmentation model), which adds multiple branches

Table 2.3: Complexity comparison between state-of-the-art DNNs and backbones.

Input size	State-of-the-art DNN		Backbone (input size scaled)	
	Model	FLOPs	Model	FLOPs
$1,920 \times 1,080$	RetinaFace [2]	9.54 G	MobileNet-v1-0.25 [28]	1.65 G
112×112	ArcFace [21]	10.13 G	ResNet [29]	0.95 G
513×513	DeepLab-v3 [22]	16.48 G	MobileNet-v2 [30]	1.54 G



(a) Offloading latency over various networks.



(b) Latency and accuracy of raw and JPEG-compressed frames.

Figure 2.1: Offloading latency of multi-DNN face identification pipeline.

to the backbone MobileNet-v2 [30] to analyze the input image in various scales.

2.2.2 Large Data Size and Compute of Each Analysis Task

Each analysis task execution requires multiple DNN inferences over high-resolution videos. Running the multi-DNN workload in low-latency is both challenging for on-device execution and cloud offloading. Figure 2.1(a) compares the end-to-end latency on-device and offloading latency for multi-DNN face identification on 1080p frame (criminal chasing scenario). We use LG V50 (Qualcomm Adreno 640 GPU) and TensorFlow-Lite for on-device inference, and a desktop server with RTX 2080 Ti GPU. We use different wireless networks: outdoor LTE (11 Mbps) and 5G (45 Mbps), indoor 802.11n (92 Mbps) and 802.11ac (292 Mbps).

First, on-device inference takes 8.6s to process a frame average 17 faces, mainly due to large number of DNN inferences. The offloading latency also remains above 0.3 s even for 802.11ac network, and increases to as high 3.4s in outdoor LTE, mainly due to the large data size (i.e., 6 MB 1080p image) to be transferred over the network. While one may think that utilizing image compression (e.g., JPEG) can reduce the network transmission latency, it comes at the cost of DNN accuracy drop as shown in Figure 2.1(b), as such compression algorithms are mainly designed to minimize the impact on human cognition [32].

2.2.3 Alternating Resource Bottleneck from Dynamic Resource Availability and Workload

Network bottleneck occurs when the available bandwidth is less than the video encoding bitrate. Compute bottleneck occurs when the DNN inference workload cannot run in real-time on the available GPUs. We observe that the bottleneck events frequently alternate across network and compute stages over time.

Study setup. We study the problem using a person analysis workload [10] in Figure 5.1, composed of YOLOv8-m [33] person detection and two ResNet-50 [29]-based face and action classifiers. Total inference latency per frame is proportional to the number of people in the scene, denoted as

$$T_{total} = T_{detect} + N_{object} \cdot T_{analysis}, \quad (2.1)$$

where T_{detect} and $T_{analysis}$ are detection and analysis latency per each of N_{object} objects in a frame, respectively. We use MOT17-11 [34] video, with 720p@30fps, 8 Mbps encoding bitrate and LTE bandwidth trace [3]. We assume the user is allocated with 1 RTX A4500 GPU, which can process the 30fps video in real-time for up to 12 objects per frame.

Results. Figure 2.2(a) and (b) indicate the occurrences of the network and compute bottlenecks in red, respectively, for a 30s analysis window. Overall, there are 5 non-overlapping bottleneck events, composed of 3 and 2 network

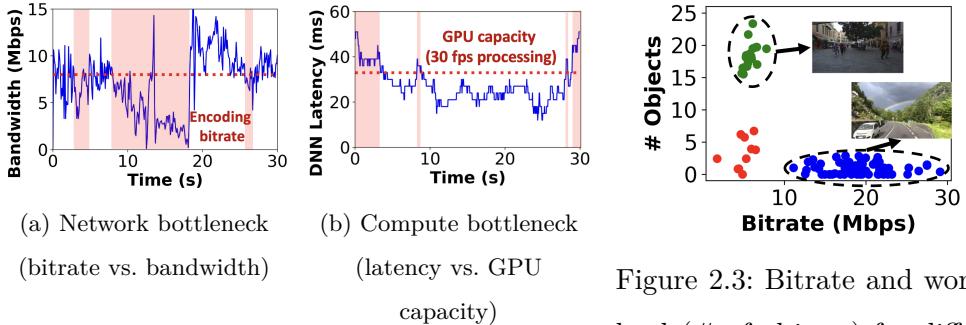


Figure 2.2: Example bottleneck (colored in red) timelines.

and compute bottlenecks, respectively, switching over time.

5.2.1.1 Why Do Bottlenecks Alternate?

We analyze that resource bottleneck alternates due to the independent fluctuation of the following factors.

Encoding bitrate and workload. Video content is highly dynamic across time and location, altering the network and compute stage resource usages (encoding bitrate and inference latency) with low correlation. Specifically, for fixed encoding parameters, the encoding bitrate of a video becomes higher for fast-changing scene content. However, this is loosely coupled with the DNN workload affected by the number of people in the scene. Figure 2.3 shows an example for different video segments from MOT [34] and self-collected YouTube videos (each dot corresponds to 1s video chunk). When a car is driving fast on a sparse highway, the dashcam video will yield a high bitrate and low inference workload (blue dots), potentially causing a network bottleneck. However, when it enters a crowded city road and moves slower, the trend will be the opposite (green dots), potentially causing a compute bottleneck.

Resource availability. Furthermore, network bandwidth fluctuates due to

Figure 2.3: Bitrate and workload (# of objects) for different scenes.

wireless channel fluctuation (e.g., due to mobility) [11, 12] and multi-user contention, independent of bitrate and workload. For example, 3GPP TS 38.306 [35] models 5G uplink throughput for highest MCS as 107 Mbps (sub-6G 100 MHz band, TDD with 5DDDSU format [36], numerology 1, single MIMO layer). With 10 contending users, each user experiences \approx 10 Mbps bandwidth, which may also drop due to channel fluctuation. Available GPU utilization ratio also fluctuates depending on other users' workload.

2.2.4 Multi-Task Resource Contention

Furthermore, running multiple DNNs and rendering tasks concurrently incurs severe resource contention, degrading performance. While this is especially serious for resource-constrained mobile devices (e.g., mobile GPUs), supporting concurrency for multi-DNN and rederring task execution is challenging due to the lack of framework and architecture support.

2.2.2.1 Multi-DNN GPU Contention

Existing mobile deep learning frameworks [18, 20, 37, 38] are mostly designed to run only a single DNN. The only way to run multiple DNNs concurrently is to launch multiple inference engine instances (e.g., TF-Lite's `Interpreter`, MACE's `MaceEngine`) on separate threads. However, multiple DNNs competing over limited mobile GPU resources incur severe contention, unexpectedly degrading the overall latency. More importantly, uncoordinated execution of multiple DNNs makes it difficult to guarantee performance for mission-critical tasks with stringent latency constraints.

To evaluate the impact of multi-DNN GPU contention on latency, we run 4 DNNs in the immersive online shopping scenario in Table 2.2 on MACE over LG V50. Figure 2.4(a) shows that with more number of DNNs contending over the mobile GPU, the inference times increase significantly compared to when only

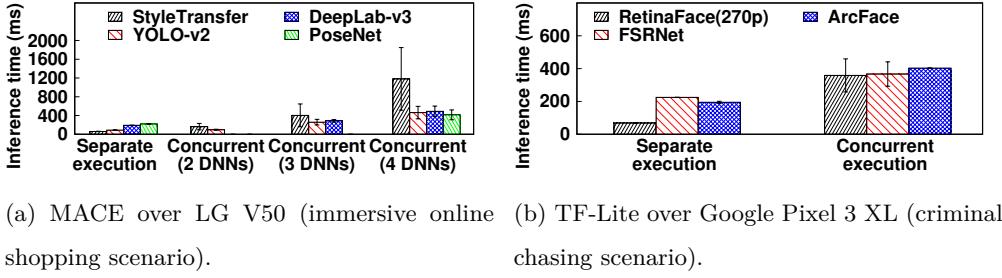


Figure 2.4: Multi-DNN GPU contention.

a single DNN is running (denoted as Separate execution). More importantly, note that the individual DNN inference times are sufficient to satisfy the app requirements (i.e., the sum of the inference times of 4 the DNNs are 560.02 ms, indicating that they can run at ≈ 2 fps when coordinated perfectly). However, the uncoordinated execution makes the performance of individual DNNs highly unstable (e.g., the latency of StyleTransfer increases from 59.93 ± 3.68 to 1181 ± 668 ms when 4 DNNs run concurrently), making it challenging to satisfy the latency requirement. We observe a similar trend in TF-Lite: Figure 2.4(b) shows that running 3 DNNs in the person identification pipeline developed in [7] incurs significant latency overhead. We further analyze the cause of the contention in Chapter 6.2.

2.2.2.2 Rendering-DNN GPU Contention

More importantly, existing frameworks only consider a single DNN running in an isolated environment (i.e., no other task contending over the mobile GPU), and are ill-suited for MR apps that require concurrent execution of rendering in presence of multiple DNNs. Figure 2.5 shows the 1080p camera frame rendering rate in presence of multiple DNNs, with the same DNN setting as in Figure 2.4. Figure 2.5(a) shows that when multiple DNNs are running, rendering frame rate drops significantly due to the similar contention between multiple DNNs

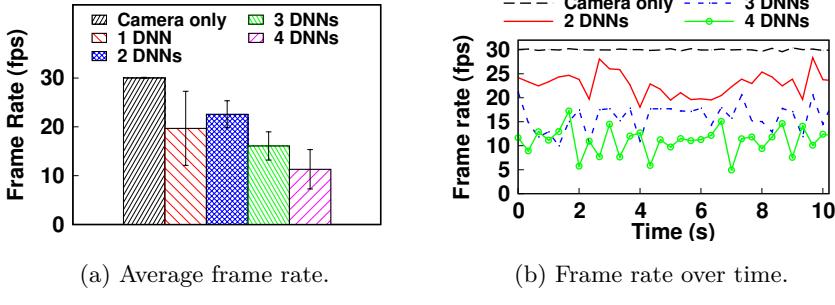


Figure 2.5: Rendering-DNN GPU contention on MACE over LG V50 (immersive online shopping scenario).

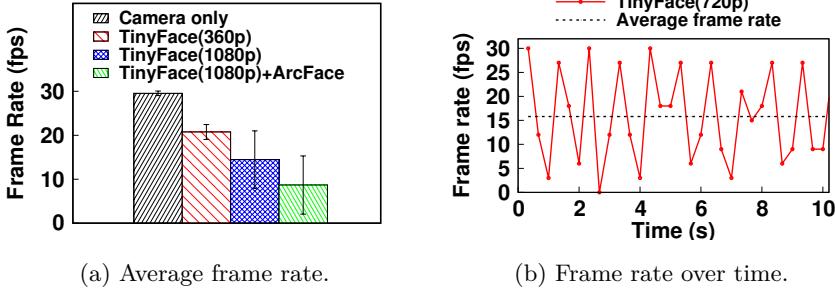


Figure 2.6: Rendering-DNN GPU contention on TF-Lite over Google Pixel 3 XL (criminal chasing scenario).

becoming as low as 11.99 fps when all 4 DNNs are running. To make matters worse, GPU contention incurs frame rate heavily fluctuating over time as shown in Figure 2.5(b), significantly degrading perceived rendering quality to users. We observe a similar trend on TF-Lite when running TinyFace [39] detector and ArcFace [21] recognizer concurrently with the rendering task (Figure 2.6).

Chapter 3

Related Work

3.1 Live Video Analytics Applications

Live video analytics enables various useful apps including traffic monitoring [40], and AR/MR [1, 7, 8]. Gabriel [41] uses cloudlets for cognitive assistance. Over-Lay [42] and MARVEL [43] utilize cloud for location-based mobile AR services. A large body of work aimed to improve the practicality of video analytics systems, including adaptation [40], model merging [44], privacy protection [45], and continual learning [46, 47]. In line with recent works, we characterize the workloads of futuristic live video analytics apps and design an edge-cloud co-operative platform to support the workload.

3.2 On-Device Systems

3.2.1 Mobile Deep Learning Frameworks

Although several frameworks have been developed from both industry [37, 38, 48, 49] and academia [18–20, 50–55], they have been mostly focused on running a single DNN in an isolated environment (i.e., no other task contending over GPU). Few studies aimed at running multiple DNNs, but are limited to be

applied for concurrent multi-DNN and rendering workload. DeepEye [16] and NestDNN [56] mainly focuses on memory optimization. DeepEye [16] parallelizes fully connected layer parameter loading and convolutional layer computation but runs only a single DNN on GPU at each time. NestDNN [56] dynamically adapts model size considering available resources but does not consider the coordination of multi-DNN inferences. Lee et al. [57] and Mainstream [58] focus on sharing weights and computations between multiple DNNs.

3.2.2 On-Device Continuous Mobile Vision

Several studies have tackled the challenge of on-device deep learning by model compression [19, 51], inference speed acceleration [18, 20, 50, 54], and model size adaptation [52, 53]. However, existing systems mostly focused on running a single DNN on downsampled images (e.g., 300×300) to analyze one or a small number of large, primary object(s) in vicinity.

For multi-task concurrency support, several studies aimed at enabling efficient GPU sharing on desktop/server GPUs, either by multiplexing multiple kernels temporally [59–61] or spatially [62–67]. Such techniques have been also applied for multi-DNN workloads [68–71]. However, they are ill-suited for mobile GPUs due to limited architecture support and memory bandwidth (see Chapter 6.3.1 for analysis).

3.3 Cloud Offloading Systems

3.3.1 Offloading for Continuous Mobile Vision

MCDNN [72] and DeepDecision [73] dynamically execute DNN on cloud or mobile based on available resources. VisualPrint [74] offloads extracted features rather than raw images to save bandwidth. Glimpse [75] tracks objects by offloading only trigger frames for detection and tracking them in the mobile. Liu

et al. [76] pipeline network transmission and DNN inference to optimize latency. However, existing systems process the input image as a whole, either on mobile or cloud at a given time; such approaches can result in significant latency in case of running complex multi-DNN pipeline.

3.3.2 Adaptive Bitrate for Live Video Analytics

A large body of works has designed adaptive bitrate techniques for live video analytics by controlling resolution [77], frame rate (frame filtering) [75, 78–84], quantization [1], and a combination of all [85]. Other works have designed RoI filtering and streaming systems [86, 87] and super-resolution-enhanced streaming pipelines [88–90]. Quantization table optimization for DNNs [32, 91] has also been studied. However, they are designed for *network-only scheduling* and cannot scale to alternating resource bottleneck scenarios.

3.3.3 ML Serving in Edge/Cloud Server

Several works aimed at high-throughput inference serving on edge/cloud servers, with content-aware adaptation [40, 92], priority-aware scheduling [5, 56, 93], caching [94–98], or multi-edge workload balancing [10]. However, they are mostly *compute-only scheduling* (assume that videos arrive at the cloud without delay), lacking scalability in network bottlenecks. For example, VideoStorm [5] allocates CPU cores across users considering resource-quality tradeoffs (profiled offline), but cannot leverage additional network resources in compute bottleneck.

3.3.4 Edge-Cloud cooperative Inference Systems

Several systems efficiently split the DNN inference workload across mobile/edge and cloud [14, 99–101]. However, they mostly focus on image processing and lack consideration for videos (e.g., how to efficiently compress the intermediate inference features of consecutive frames). Furthermore, they mostly assume single

task scenarios and lack consideration for multi-DNN and rendering concurrency support.

3.4 Tiny ML/Efficient Deep Learning

A large body of works aimed at DNN compression for resource-efficient deep learning in mobile/edge devices. They have leveraged various techniques including lightweight model architecture design [28, 33], weight pruning [102], quantization [103], combination of both [104, 105], hardware-aware model adaptation [106], and neural architecture search [107, 108]. Such optimization techniques can also be leveraged in our platform for resource-efficiency.

Chapter 4

EagleEye: AR-based Person Identification in Crowded Urban Spaces

4.1 Introduction

In this Chapter, we design EagleEye, a system for content-aware adaptation and edge-cloud collaborative execution in live video analytics. We take the AR person finding application as the representative multi-DNN live video analytics workload. Imagine a parent looking for her/his missing child in a highly crowded square. In many cases, a swarm of people in front of her/his eyes will quickly overload cognitive abilities; our motivational study shows that it takes ≈ 16 seconds to locate a person in a crowded scene (See Chapter 4.3 for details). An Augmented Reality (AR)-based service with smart glasses or a smartphone will be extremely helpful if it can capture the large crowd from distance and pinpoint the missing child in real-time (Figure 4.1). Despite recent advances in person identification techniques using various features such as face [21, 109, 110], gait [111, 112] or sound [113, 114], fast and accurate person identification in crowded urban spaces remains a highly challenging problem.



Figure 4.1: Example usage scenario of EagleEye: parent finding a missing child.
More examples in Chapter 4.2.

EagleEye is a AR-based system to identify missing person(s) in large, crowded urban spaces. It continuously captures the image stream of the place using commodity mobile cameras, identifies person(s) of interests, and shows where the target is in the scene in (soft) real-time. EagleEye not only shows a good example of future AR applications based on real-time analysis of complex scenes, but also characterizes the workload of future multi-DNN mobile deep learning systems.

Designing EagleEye involves critical technical challenges for both identification accuracy and latency.

- **Recognition accuracy.** Compared to prior systems [115–117] that aim at identifying 1 or 2 faces in close vicinity (e.g., engaged in a conversation), the key challenge in building EagleEye is accurately detecting and recognizing distant small faces. In crowded spaces, individual faces often appear very small, with facial details blurred out. Recent Deep Neural Network (DNN)-based face recognition has shown remarkable progress in accurately identifying faces under various unconstrained settings [21, 118, 119] (e.g., variations in pose, occlusion, or illumination). However, the state-of-the-art techniques still fail to provide robust performance for Low-Resolution (LR) faces. Our study shows that Equal Error Rate, the value in the ROC curve where false acceptance and false rejec-

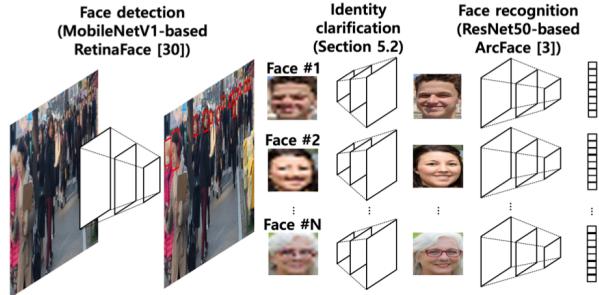


Figure 4.2: Multi-DNN face identification pipeline.

tion rates are identical, of the state-of-the-art DNN [21] grows from 9% to 27% when resolution drops from 112×112 to 14×14 (Chapter 4.3).

- **Identification latency.** More importantly, it is challenging to analyze a crowded scene in (soft) real-time to allow users to sweep large spaces quickly. EagleEye imposes unique challenges compared to recent DNN-based continuous mobile vision systems [16, 18–20, 72, 73, 76]. Firstly, as shown in Figure 4.2, EagleEye requires running a series of complex DNNs multiple times for a single scene: face detection network once over a scene, our resolution enhancing network (introduced in Chapter 4.5.2) and face recognition network per each face. This is very different from prior systems that run a single DNN only once over a scene. Secondly, each DNN is highly complex to achieve high accuracy, incurring significant latency. Face detectors employ feature pyramid [31] which upsamples features in latter layers and adds up to earlier layers to detect small faces. Also, state-of-the-art recognizers are heavy ResNet-based. Finally, prior work mostly downsample the input frames (e.g., 300×300 [120]) to reduce complexity (this was possible as they analyze a small number of large, primary objects in vicinity). However, EagleEye should run the identification pipeline on high-resolution frames to detect a large number of distant faces that appear very small.

It is highly challenging to run a complex multi-DNN pipeline over high-resolution images in real-time. It is not even trivial to simply port state-of-the-art DNNs to mobile deep learning frameworks (e.g., TensorFlow-Lite) due to the limited number of supported operations. The challenge aggravates considering the execution latency. For instance, a lightweight MobileNet [28] can only process two 1080p frames per second on high-end mobile GPU (Table 4.1). Naive execution of `EagleEye`'s entire pipeline takes 14 seconds for a scene with 30 faces (Figure 4.5). We can consider multithreading or offloading, but they are not also straightforward to apply. Multithreading degrades performance due to resource contention over limited mobile resources (e.g. GPU, CPU, memory). Also, 3G/LTE network with low bandwidth is likely the only wireless network available in crowded outdoor environments, making offloading non-trivial.

To tackle the challenges, we design and develop a suite of novel techniques and adopt them in `EagleEye`.

- **Identity clarification network.** We first design a novel end-to-end face identification pipeline to identify small faces accurately. Our key idea is to add *Identity-Clarification Network (ICN)* on conventional 2-step pipeline (detection-recognition) to recover missing facial details in LR faces, thus resulting in a 3-step pipeline (detection-clarification-recognition as shown in Figure 4.2).¹ ICN adopts a state-of-the-art image super-resolution network as the baseline and innovates it with specialized training loss functions to enhance LR faces for accurate recognition; note that prior super-resolution networks focus on generating perceptually natural images and fail to preserve identities, making them ill-suited for recognition [121] (see Chapter 4.5). Also, ICN enables identity-preserving reconstruction using reference images (*probes*) of the target, com-

¹Instead of adding ICN to the conventional 2-step pipeline, integrating a resolution up-sampler in the feature encoder of a face recognition model and training it on low-resolution faces is also feasible. However, training and running the new model would still incur similar accuracy-latency challenges.

monly available in our scenarios (e.g., photos of children provided by parents). We observe that the complexity of LR face recognition results from accepting positive identities rather than denying negative identities (see Chapter 4.5.2 for details). Thus, biasing ICN on the target improves LR face recognition accuracy with only a small increase in false positives. Overall, our ICN-enabled pipeline improves true positives by 78% with 14% false positives, against the 2-step identification pipeline.

- **Multi-DNN execution pipeline.** Our workload (i.e., running a series of DNNs multiple times on high-resolution images) requires a differentiated strategy to optimize the heavy computation. We develop a runtime system with *Content-Adaptive Parallel Execution* to run a multi-DNN face identification pipeline at low latency. The key idea behind this approach is to divide the high-resolution image into multiple sub-regions and selectively enable different components in the pipeline, depending on the content. For instance, ICN is only applied to a region with LR faces while the entire pipeline is not executed for a background region with no faces. Furthermore, we exploit the spatial independence of face recognition workload (i.e., identifying faces in different sub-regions does not have dependency) to parallelize and pipeline the execution on heterogeneous processors on the mobile and cloud. Overall, our technique accelerates the latency by $9.07\times$ with only 108 KBytes of data offloaded.

Our major contributions are summarized as follows:

- To the best of our knowledge, this is the first end-to-end mobile system that provides accurate and low-latency person identification in crowded urban spaces.
- We design a novel face identification pipeline capable of accurately identifying small faces in crowded spaces. By employing Identity Clarification Network to recover facial details of LR faces, we enhance true positives by 78% with 14% false positives.

- We design a runtime system to handle the unique workload of EagleEye (i.e., processing high-resolution images with multiple DNNs for complex scene analysis). We believe this will be an unexplored common workload for many mobile/wearable-based continuous vision applications. We utilize a suite of techniques to minimize the end-to-end latency to as low as 946 ms ($9.07\times$ faster than naive execution).
- We conduct extensive controlled and in-the-wild study (with real implementations and various datasets), validating the effectiveness of our proposed system.

4.2 Motivating Scenarios

Finding a missing child. In crowded squares or amusement parks, there are many cases where a parent loses track of her/his child. In such incidents, it is difficult to find the missing child with naked eyes since she/he becomes cognitively overloaded to identify many people in vicinity. EagleEye can help the parent: by sweeping the mobile device to capture the space from distance, it can help quickly pinpoint possible faces and narrow down a specific area to search, so that the parent can find the child before the child moves to a different place. Similarly, police officers can use EagleEye to chase criminals in crowded malls, streets, squares, etc.

Children counting in field trips. Teachers in kindergarten regularly take children out for field trips to catch educationally-depicting behaviors hardly captured in classroom settings. However, in reality, teachers spend most of the time counting children to make sure they are in place. EagleEye can be of extensive use to reduce the cognitive burden for the teachers so that they can focus on the original goal.

Social services for familiar strangers. EagleEye can be used to build an

interesting social service to connect people. For example, it can be used to identify familiar strangers (people whom we met in the past but do not remember the details) to help with interaction; a person attending a social event can use **EagleEye** to identify them and get an early heads-up before they are in close proximity to avoid embarrassing moments.

4.3 Preliminary Studies

To motivate **EagleEye**, we first conduct a few studies to verify (1) how quickly humans can identify face(s) in crowded urban spaces and (2) whether it is feasible in terms of accuracy and speed to employ DNN-based face recognition algorithms to aid the humans' cognitive abilities.

4.3.1 How Fast Can Humans Identify Faces?

Prior studies report that it takes for humans about 700 ms to detect a face in a scene [122], and about 1 second to recognize the identity of a single face image [123]. We extend the experiments to study how long it takes to identify target(s) in crowded scenes. We first recruit 6 college students (5 males and 1 female, age 24-28) as subjects for dataset collection, and take videos of them blending inside the crowd in various urban spaces including college campus, downtown streets, and subway stations. Next, we recruit 11 students (10 males and 1 female, age 24-32) who are of mutual acquaintances with the subjects (denoted as *Familiar*), and 14 other students (12 males and 2 females, age 20-26) who have never seen the subjects before (denoted as *Unfamiliar*).

In the experiments, the participants are seated in front of the screen with a similar setup as in [122]. Each participant is first shown faces of 1 to 3 target identities. Afterwards, a scene image (1080p resolution) is shown, in which target(s) may or may not exist. The participant clicks the location in the scene

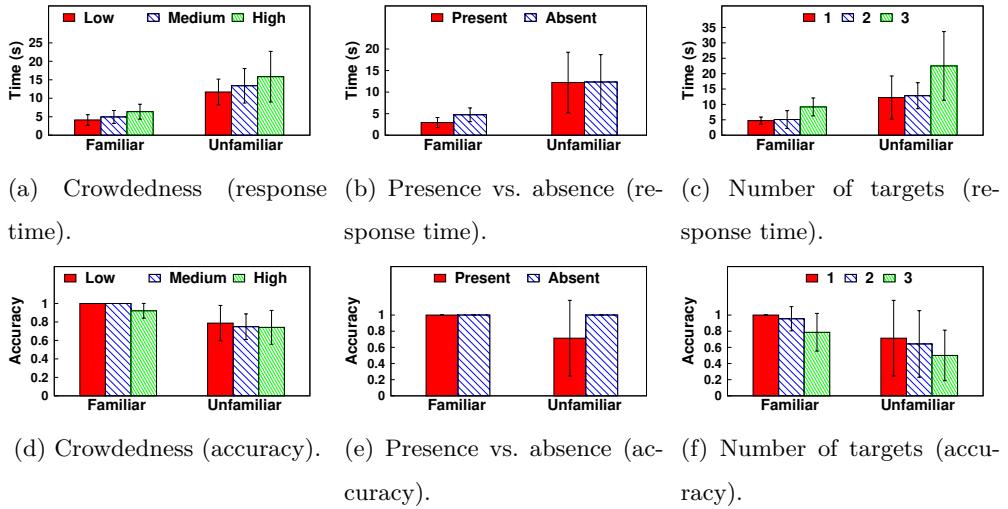


Figure 4.3: Human cognitive abilities on identifying faces in crowded scenes: response time and accuracy.

where she/he finds each target. Response time is measured as the duration between when the scene is displayed and when the participant finishes identifying all targets. The scenes are classified into three levels of crowdedness (examples are shown in Figure 4.16): i) *Low* (less than 10 people in close distance with face sizes at least 30×30 pixels), ii) *High* (more than 20 people with face sizes smaller than 14×14), and iii) *Medium* (between *Low* and *High*). Each participant is shown 5 scenes per each category (15 in total) and was asked to be as precise as possible.

Figure 4.3 shows the response time/accuracy results. Our experimental results are summarized as follows (unless specified, the reported results are on *High* scenes):

- Overall, it takes 6.37 and 15.83 seconds on average to identify familiar and unfamiliar faces in crowded scenes (standard deviation: 2.02 and 6.87 seconds), respectively, showing noticeable cognitive loads.
- It takes longer to identify unfamiliar faces than familiar ones.
- Not only does it take longer to identify a target in more crowded scenes, but

the accuracy also drops (Figures 4.3(a) and (d)).

- Especially for the *Familiar* group, it takes longer to confirm the absence of target than presence. (Figures 4.3(b) and (e)). We observe that it is because when participants fail to locate the target in the scene, they start looking over again multiple times to confirm their decision.
- It takes longer to identify multiple targets, and accuracy drops as well (Figures 4.3(c) and (f)).

The above results clearly show the human’s vulnerability to cognitive overload. While the study was designed as identifying the target person(s) in a scene image for controllability of the experiment, we conjecture that the cognitive overload will be greater in real-world settings where the scene does not fit into a single view.

4.3.2 How Accurate Can DNNs Identify Faces?

Faces in crowded spaces captured from a distance experience high variations in pose, occlusion, illumination, and resolution, making accurate recognition very challenging. While prior algorithms have achieved robust performance (e.g., over 90% accuracy) for the first three [21, 118, 119], the Low-Resolution (LR) face recognition problem has not been fully studied yet.

We conduct a study to analyze the difficulty of LR face recognition. We first train ResNet50 with ArcFace loss [21] on MS1M dataset [124], and test performance on 50 identities in VGGFace2 [125] testset (50 images per identity). Figure 4.4 shows that verification (determining whether two faces match or not) accuracy drops significantly as resolution decreases. Equal Error Rate (EER), the value in the ROC curve where false acceptance and false rejection rate are identical, grows as high as 0.27 when the resolution is 14×14 .

For further analysis, we run a small study with 8 identities in VGGFace2 [125] testset. We train ResNet50 [29] with 2-dimensional output features using SphereFace

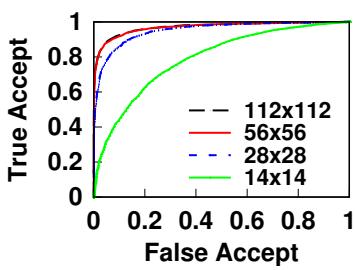


Figure 4.4: Face verification accuracy.

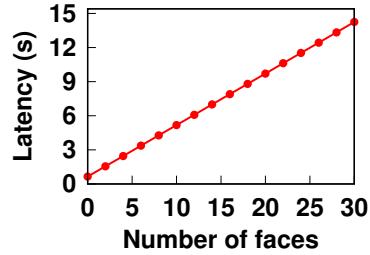


Figure 4.5: Latency of face identification pipeline.

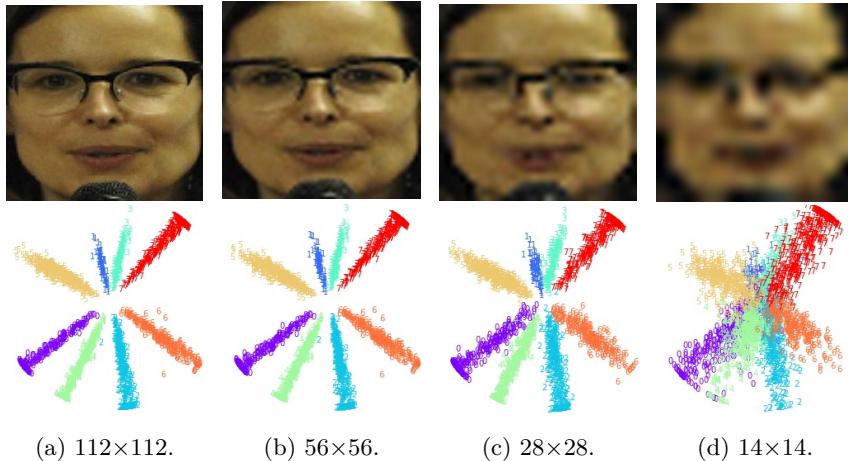


Figure 4.6: Feature map visualization for varying resolutions (points with same color represents same identity).

loss [109]. Figure 4.6 visualizes the trained features for varying resolutions, where the points with the same color represent the same identity. We observe that when the resolution is high (e.g., 112×112), features for each identity form non-overlapping sharp clusters. However, as resolution drops, clusters become wider and start to overlap with each other, becoming indistinguishable.

Table 4.1: Inference time of DNNs with TensorFlow-Lite running on LG V50 (Qualcomm Adreno 640 GPU).

Input size	Model	
	MobileNetV1 [28] (Classification)	YOLO-v2 [23] (Detection)
224×224	24 ms	357 ms
640×360	55 ms	1,477 ms
1,280×720	209 ms	5,009 ms
1,920×1,080	452 ms	9,367 ms

Table 4.2: Complexity and latency of component DNNs. FLOPs are measured with *tf.profiler.profile()* function.

Task	Model	FLOPs	Inference time
Face detection	RetinaFace [2] (MobileNetV1-based)	9.54 G	648 ms per 1080p image
Identity clarification	Ours (Chapter 4.5.2)	15.84 G	166 ms per 14×14 face
Face recognition	ArcFace [21] (ResNet50-based)	10.21 G	287 ms per 112×112 face

4.3.3 How Fast Can DNNs Identify Faces?

Conventional face identification pipelines operate in a 2-step manner (i.e., face detection on the image and face recognition on each detected face sequentially). In our scenarios, both steps require significant computation. First, the detection network should run on a high-resolution frame to detect distant faces that appear very small. In such settings, providing real-time performance is challenging; Table 4.1 shows that YOLOv2 [23], one of the fastest networks that can be used for face detection, takes more than 9 seconds to process a 1080p frame. Second, recognition latency increases proportionally to the number of faces, which can be very large in crowded scenes. Figure 4.5 shows that naively running the state-of-the-art multi-DNN face identification pipeline composed of

DNNs summarized in Table 4.2² takes more than 14 seconds to process a scene with 30 faces even on a high-end LG V50 with Qualcomm Adreno 640 GPU.

4.3.4 Summary

In crowded spaces, humans become cognitively overloaded, clearly necessitating the need for a system to aid their abilities. However, DNN-based face recognition algorithms cannot be applied directly as they fail to identify LR faces accurately, and naive execution incurs significant latency.

4.4 EagleEye: System Overview

4.4.1 Design Considerations

High recognition accuracy. Our primary objective is to design a face identification pipeline capable of accurately identifying target(s) in crowded spaces, even when he/she appears very small.

Soft Real-time performance. While enabling an accurate face identification pipeline, our goal is to provide soft real-time performance (e.g., 1 fps) for application usability. We aim to devise techniques to optimize various latency components in the end-to-end system while incurring a minimum loss in recognition accuracy.

Use of commodity mobile camera. We aim at achieving high accuracy using frames captured by cameras of commodity smartphones or wearable glasses (e.g., 1080p frames at 30 fps [127]). If cameras with higher resolution or optical zoom-in are available, our approach can help cover a more extensive search area.

²These are the state-of-the-art not only in terms of accuracy but also in terms of complexity. For face detectors, comparable networks are heavy VGG16 [126] or ResNet101 [39]-based. Recent face recognizers are based on 64-layered ResNet [109, 110].

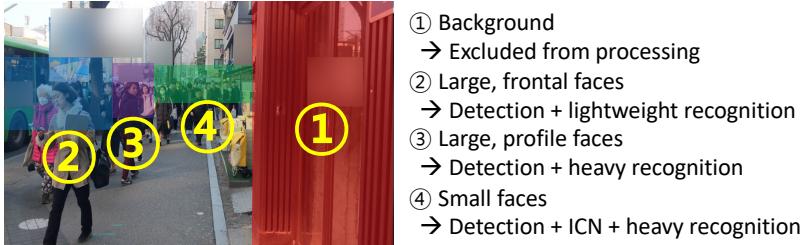


Figure 4.7: Operation of EagleEye in a nutshell.

Minimal use of offloading. In our common use cases (i.e., a moving user in crowded outdoor environments), we assume that the availability of edge servers and Wi-Fi connectivity are limited. For robust performance, we aim to minimize the amount of data offloaded to the cloud and run most of the computation on local.

4.4.2 Operational Flow

Figure 4.7 shows the nutshell operation of EagleEye: given a crowded scene image, we adaptively process each region with different pipelines depending on the content. For background regions, we do not run any DNN. For non-background regions, we run face detection and adaptively select the latter part of the pipeline to process each detected face based on different variations: i) large, frontal faces (which are very easy to recognize) are processed with a lightweight recognition network, ii) large, profile faces (whose resolutions are sufficient but pose variations make recognition difficult) are processed with a heavy recognition network, and iii) small faces are first processed with *Identity Clarification Network* (which enhances resolution of LR faces for accurate recognition) and then with heavy recognition network. Finally, exploiting the spatial independence of the task, we process each region and face in parallel on heterogeneous processors on mobile and cloud.

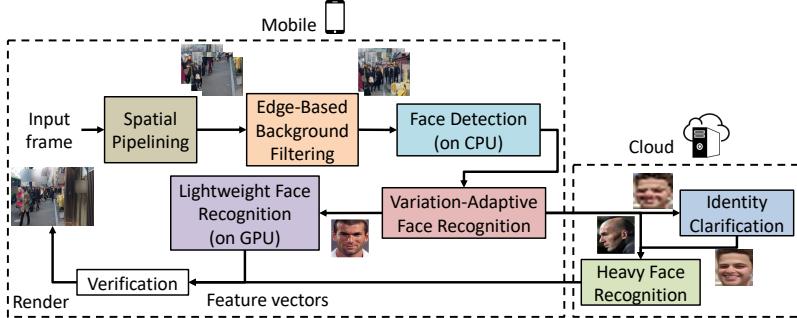


Figure 4.8: EagleEye system overview.

Figure 4.8 shows the operational flow of EagleEye. We employ *Content-Adaptive Parallel Execution* to run the complex multi-DNN face identification pipeline at low latency using heterogeneous processors on mobile and cloud. Given an input frame, *Spatial Pipelining* first divides it into spatial blocks, so that each block can be processed in a pipelined and parallel manner. Afterwards, *Edge-Based Background Filtering* rules out background blocks with edge intensity lower than a threshold. For the remaining blocks, we detect faces on the mobile CPU. Each detected face is scheduled to a different pipeline by *Variation-Adaptive Face recognition*. Large, frontal faces are processed by lightweight recognition network running on mobile GPU. The rest is offloaded to the cloud, where large, profile faces are processed by heavy recognition network, and small faces are processed by ICN followed by heavy recognition network.

4.5 Identity Clarification-Enabled Face Identification Pipeline

We detail our novel 3-step face identification pipeline. It operates as shown in Figure 4.2: i) detect faces in the scene, ii) enhance each LR face with ICN, and iii) extract feature vectors for each face with recognition network.



Figure 4.9: GANs reconstruct realistic faces, but fail to preserve the face identity.

4.5.1 Face Detection

The first step of our pipeline is face detection. The detection network should be accurate in detecting small faces, since faces missed in this step would lose the chance of being identified at all. At the same time, it should be lightweight so that it can run in (soft) real-time. We experiment various state-of-the-art DNNs and select RetinaFace detector [2] with MobileNetV1 [28] backbone for the following reasons: i) it adopts context module which has been proven very effective in detecting small faces [126, 128], and ii) it is the fastest among the state-of-the-art group due to its lightweight backbone network (others are heavy VGG16-based [126] or ResNet101-based [39]).

4.5.2 Identity Clarification Network

LR faces lack details crucial for identification. To enhance recognition accuracy, we design *ICN*, which enhances the resolution of LR faces using Generative Adversarial Network (GAN). As conventional GANs reconstruct faces with significant distortion from the original identity (Figure 4.9), we adapt GAN to reconstruct identity-preserving faces by using various loss functions, as well as a specialized training methodology (*Identity-Specific Fine-Tuning*).

Network Architecture. Figure 4.10 shows the overview of ICN. For generator G , we adopt Residual block [29]-based architecture similar to FSRNet [129]

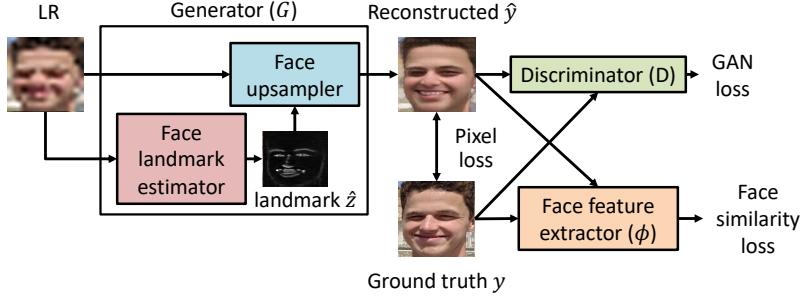


Figure 4.10: Identity Clarification Network: overview.

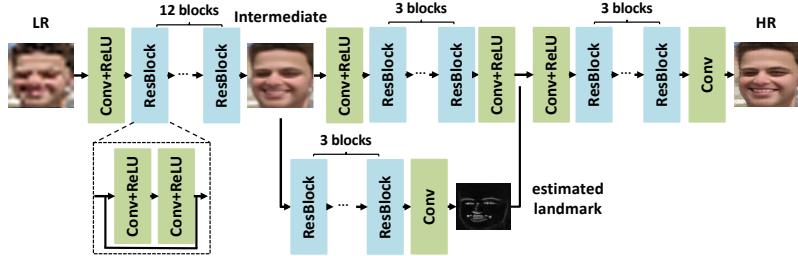


Figure 4.11: Generator network architecture.

as shown in Figure 4.11, which has shown high reconstruction performance. Furthermore, we employ anti-aliasing convolutional and pooling layers [130] to improve robustness to pixel misalignment in face detection and cropping process. We employ various additional networks and loss functions to train ICN to preserve identity as follows.

Following the convention in super-resolution [131, 132], the generator is trained to minimize the pixel-wise L2 loss between the reconstructed face and the ground truth,

$$L_{pixel} = \frac{1}{HW} \sum_{i=1}^H \sum_{j=1}^W \left(\|y_{i,j} - \tilde{y}_{i,j}\|^2 + \|y_{i,j} - \hat{y}_{i,j}\|^2 \right), \quad (4.1)$$

where H, W are height and width, \tilde{y} and \hat{y} are the intermediate and final High-Resolution (HR) face in Figure 4.11, respectively, and y is the ground truth.

As reconstructing HR faces is very challenging, recent studies have shown

that employing a facial landmark estimation network to guide the reconstruction process yields superior performance [129, 133]. We adopt the approach to estimate facial landmarks from the intermediate HR face instead of directly from the LR face. The facial landmark estimation network is trained to minimize the MSE between estimated and ground truth landmarks,

$$L_{landmark} = \frac{1}{N} \sum_{n=1}^N \sum_{i,j} \|z_{i,j}^n - \hat{z}_{i,j}^n\|^2, \quad (4.2)$$

where $\hat{z}_{i,j}^n$ is the estimated heatmap of the n -th landmark at pixel (i, j) and z is the ground truth.

Recent studies have shown that GAN [134] plays an important role in reconstructing realistic images. We employ WGAN-GP [135] for improved training stability, whose loss is defined as:

$$L_{GAN} = -D(\hat{y}) = -D(G(x)), \quad (4.3)$$

where $G(x)$ denotes the HR face reconstructed by the generator, and D denotes the discriminator that classifies whether the reconstructed face looks real or not, which is trained by minimizing the following loss function (refer to the original paper [135] for details),

$$L_{Discriminator} = D(\hat{y}) - D(y) + \lambda (\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2. \quad (4.4)$$

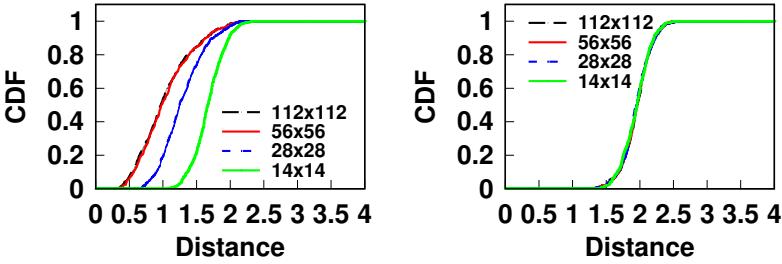
We also enforce the reconstructed face to have similar features with the ground truth by minimizing the face similarity loss

$$L_{face} = \frac{1}{d} \|\psi(y) - \psi(\hat{y})\|^2, \quad (4.5)$$

where $\psi(\cdot)$ denotes d -dimensional feature vector extracted by the VGG16 network trained on ImageNet [136].

Finally, the above loss functions are combined as a weighted sum and minimized in the training process,

$$L_{total} = L_{pixel} + 50 \cdot L_{landmark} + 0.1 \cdot L_{GAN} + 0.001 \cdot L_{face}. \quad (4.6)$$



(a) Same identity pair.

(b) Different identity pair.

Figure 4.12: CDF of face distances for varying resolutions.

Identity-Specific Fine-Tuning. Baseline ICN aims to adapt conventional GANs to overcome their limitation (i.e., reconstructing perceptually realistic faces at the cost of significant distortion from the ground truth). However, we notice that it still often reconstructs faces with distorted identity from the original. Accordingly, we need another step to employ ICN for our purpose of accurate recognition.

Before introducing our approach, we further dig deeper into the LR face recognition problem. Figure 4.12 shows that as resolution decreases, L2 distance between features of faces with the same identity increases significantly, whereas those of different identities remain identical. In other words, the difficulty of LR face recognition comes from the hardship of accepting positive pair of faces, rather than denying negative pairs. Therefore, LR face recognition accuracy can be enhanced if we can bring back the features of faces with the same identity close to each other.

To this end, we develop *Identity-Specific Fine-Tuning* to re-train ICN with reference images (*probes*) of the target, which is commonly available in our target scenarios (e.g., photos of children provided by parents). Such re-training process enables ICN to instill the facial details of the target into the input LR face, thus making it easier to recognize when a LR face of target identity is captured. While such biasing may also increase false positives caused by

LR faces that do not match the target identity pulled towards the probes, we observe that such cases only occur for ones that are very close to the target in feature space, thus yielding gain in true positives outweighing false positives (78% vs. 14% as shown in Chapter 4.8.3).

Probe Requirements. To fine-tune the ICN to instill facial details of the target, Identity-Specific Fine-Tuning requires probe images with rich facial details. As an initial study we collect the probes with high-resolution, and leave detailed analysis of the impact of the composition of probes (e.g., pose or occlusion) as future work.

Data Augmentation. To diversify the probes as well as boost robustness to various real-world degradation, we also utilize the following augmentation techniques:

- **Illumination.** Change value (V) component in HSV color space.
- **Blur.** Apply Gaussian blur with varying kernel sizes.
- **Noise.** Add Gaussian noise with varying variance.
- **Flip.** Apply horizontal flip.
- **Downsampling.** Resize with different downsampling kernels. (e.g., bicubic, nearest neighbor).

Scalability. Finally, the overhead of fine-tuning the baseline ICN pre-trained on a large-scale face dataset to a specific target identity is not significant (e.g., takes about 20 minutes on a single NVIDIA GTX 2080Ti GPU). Thus, we expect it can be flexibly re-trained at deployment as the target changes.

4.5.3 Face Recognition and Service Provision

At the final stage, state-of-the-art ResNet50-based ArcFace [21] runs on each face to extract 512-dimensional feature vector, which is compared to that of the target probes. Those with distance below the threshold are highlighted on the screen so that the user can take further actions. To compensate for possible

motion between the image capture and output rendering (about 1 second as our evaluation shows), we can employ motion tracking to shift the bounding boxes using approaches used in prior detection systems [75, 76].

4.6 Real-Time Multi-DNN Execution

We detail our runtime system to execute the multi-DNN face identification pipeline at low latency. We start with workload characterization by identifying the sources of latency, followed by our proposed *Content-Adaptive Parallel Execution*.

4.6.1 Workload Characterization

Sequential Execution of Multiple DNNs. Identifying target person(s) in a crowded scene requires a sequential execution of multiple complex DNNs (i.e., face detection, identity clarification, and recognition) whose individual complexities are summarized in Table 4.2.

High-Resolution Input. Conventional object detection networks downsample the input images to reduce complexity (e.g., 416×416 [23] or 300×300 [120]). However, in our case, the input image size should be retained large (e.g., 1080p), so that small faces have enough pixels to be detected. As the complexity of DNN inference grows proportionally to the image size, latency becomes significant when processing such high-resolution images.

Repetitive Execution for Each Face. ICN and recognition network must repeatedly run for each face detected by the face detection network. The latency increases proportionally to the number of faces in the scene, which becomes significant in crowded spaces.

4.6.2 Content-Adaptive Parallel Execution

4.6.2.1 Optimization Strategies

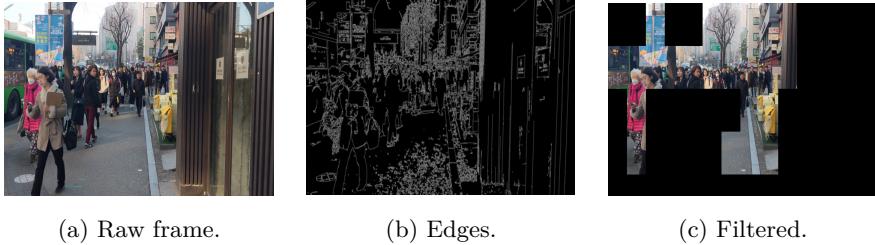
Content-Adaptive Pipeline Selection. We adaptively process each region of the image with different pipelines depending on the content. This helps optimize the latency incurred when processing a large number of faces, while maintaining high recognition accuracy.

Spatial Independence and Parallelism. Identifying faces in different regions of the image is spatially independent. Furthermore, recognizing each detected face can be executed simultaneously. To take full advantage of such opportunities for parallelism, we divide the image into spatial blocks and process them in a pipelined and parallel manner using heterogeneous processors on mobile and cloud. This helps optimizing the latency of multi-DNN execution on high-resolution images.

4.6.2.2 Content-Adaptive Pipeline Selection

We develop techniques to optimize the latency of complex multi-DNN face identification pipeline execution while maintaining high accuracy. Specifically, *Edge-Based Background Filtering* rules out background regions where faces do not exist at all. *Variation-Adaptive Face Recognition* selects different recognition pipelines depending on recognition difficulty.

Edge-Based Background Filtering. Running face detection on regions where faces do not exist at all (e.g., background) is a wasteful computation. To mitigate the problem, we use edges in the image to rule out such regions before running the identification pipeline. Specifically, given a frame as shown in Figure 4.13(a), we detect edges as in Figure 4.13(b), filter out blocks with edge intensity below a threshold as depicted in Figure 4.13(c), and run face detection only on the remaining blocks. Note that edge detectors are extremely



(a) Raw frame.

(b) Edges.

(c) Filtered.

Figure 4.13: Edge-based background filtering.

lightweight, especially considering that we can even detect edges on downsampled images. For example, the time complexity of Canny edge detector [137] for $H \times W$ frame is $O(HW \cdot \log(HW))$, and it runs in less than 2 ms for 360p frame on LG V50. Thus, its overhead is minimal even when the edge detection is not effective for some scenes having full of objects and no background regions.

Variation-Adaptive Face Recognition. State-of-the-art recognition networks are designed very complex (e.g., heavy ResNet backbone with a large number of batch normalization layers) to accurately identify faces even under high variations in pose, illumination, etc. However, employing such heavy networks for faces in ideal conditions is an overkill. For example, MobileFaceNet [138] and ResNet50-based ArcFace [21] achieve comparable accuracy on LFW [139] dataset composed of large, frontal faces (98.9% vs. 99.3%), whereas inference time differs by more than 20 \times (14 ms vs. 287 ms). Therefore, we aim to optimize latency by adaptively processing each face depending on its variation (i.e., recognition difficulty).

Figure 4.14 depicts our Variation-Adaptive Face Recognition, which utilizes the size of bounding box and 5 face landmarks detected by RetinaFace [2] detector (other state-of-the-art face detectors also provide 5 landmarks as outputs). First, small faces are processed by ICN and then by ResNet50-based ArcFace [21]. For large faces, we estimate the pose using the detected landmarks; for example, if the angle between the line connected by points (2, 3)

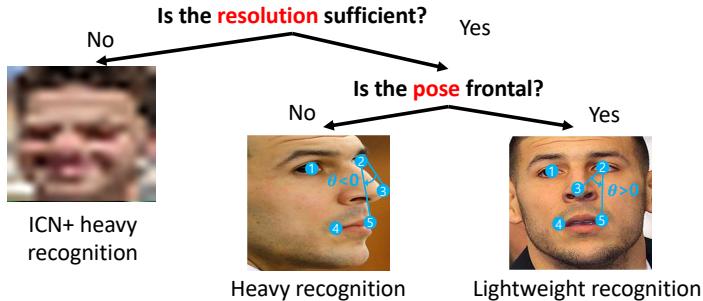


Figure 4.14: Variation-Adaptive Face Recognition.

and (2, 5) measured in counterclockwise direction is negative, we can tell that the face is looking to the right. As faces with pose variations are difficult to accurately identify, they are also processed by ResNet50-based ArcFace (ICN is not needed here as resolution is already sufficient). The remaining faces (large and frontal) which are easy to identify are processed by MobileFaceNet [138].

We finally note that the content-adaptive inference can also be integrated inside the face recognition model (e.g., through learnable input filtering [140] or early-exit [14] techniques). While our Variation-Adaptive Face Recognition technique uses lightweight features (resolution and landmarks) to estimate recognition difficulty, such model-integrated approaches can make the adaptation more fine-grained and accurate.

4.6.2.3 Execution Planning

We optimize the latency of multi-DNN face identification pipeline by scheduling each component DNN inference execution to the most suitable processor on mobile and cloud.

Offloading Decision. As our target scenarios assume crowded outdoor environments with congested 3G/LTE network, offloading high-resolution images for detection is impractical; instead, we offload only the detected faces. Specifi-

Algorithm 1 Combined operational flow of EagleEye

```
1: while application is running do
2:    $Result \leftarrow \{\}$ 
3:    $Frame \leftarrow acquireFrameFromCamera()$ 
4:    $Edges \leftarrow EdgeDetector(Frame)$ 
5:    $NonBackground \leftarrow BackgroundFilter(Edges)$  Block in NonBackground
6:    $Faces \leftarrow FaceDetection(Block)$  face in Faces
7:    $Result \leftarrow Result \cup AdaptiveFaceRecognition(face)$ 
8: end while
```

cally, LR faces are suitable for offloading, as their data sizes are very small (e.g., 14×14 pixels) whereas the required computation (i.e., ICN and heavy recognition) incurs significant latency on mobile (e.g., 166+287 ms). We also offload large, profile faces, and leave only the large, frontal faces to be processed by lightweight recognition on mobile.

Mobile Processor Mapping. The mobile needs to run both detection and lightweight recognition. However, simply multithreading the execution on GPU does not help optimize latency, as mobile GPUs lack preemptive multitasking support. Therefore, we utilize heterogeneous processors (CPU and GPU) to parallelize the execution. As dynamically switching the mapping over time is challenging due to high latency overhead of loading DNN on mobile GPUs (e.g., 2 seconds for 118 MB ResNet50-based ArcFace [21] on LG V50 with TensorFlow-Lite), we statically run detection on CPU and recognition on GPU considering the following aspects:

- **Memory I/O.** Running face detection on GPU requires high-resolution images loaded onto GPU memory, and output feature maps from different stages in the feature pyramid (whose size is proportional to the input image size) copied back to CPU to be post-processed to bounding boxes. Considering memory overhead, it is more suitable to run face recognition on GPU whose input/output are small-sized faces and 1D feature vectors.

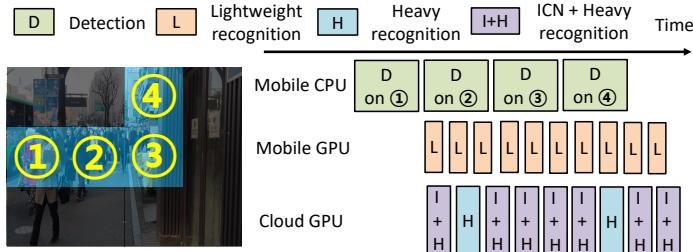


Figure 4.15: Spatial Pipelining on heterogeneous processors.

- **Inference time.** Besides, we observe that the inference speed slowdown of RetinaFace detector running on CPU is $1.22\times$ (648 vs. 793 ms), whereas it is $2.07\times$ for MobileNetV1-based ArcFace recognizer (14 vs. 29 ms). Therefore, running detection on CPU and recognition on GPU is more feasible to optimize overall latency, especially when the number of faces is large.

4.6.2.4 Spatial Pipelining

To further optimize the latency, we exploit the spatial independence of the workload by processing each image sub-block in a pipelined and parallel manner. As depicted in Figure 4.15, given non-background blocks in a scene, we detect faces in one block on mobile CPU, while simultaneously processing faces detected in another block on mobile and cloud GPU.

Note that we need to divide the image into blocks in an overlapping manner with padding, so as to prevent faces from being split across different blocks (and thereby failing to be detected). While fine dividing increases the chance of higher parallelism, it also increases the computational overhead due to padding. Based on our empirical evaluation on such tradeoff in Chapter 4.8.4, we divide an image into 4x4 blocks.

4.6.2.5 Putting Things Together

Algorithm 1 summarizes the combined operational flow. Upon acquiring a frame from the camera, we detect edges (line 4) and filter out background (line 5). For non-background blocks (line 5), we run face detector on CPU (line 6) and process each face adaptively in mobile or cloud GPU (lines 6–7) in a pipelined and parallel manner. Finally, the recognition result is rendered on the screen.

4.7 Implementation

Mobile. We implement the mobile side of **EagleEye** on two commodity smartphones running on Android 9.0.0: LG V50 with Qualcomm Snapdragon 855 and Adreno 640 GPU and Google Pixel 3 XL with Qualcomm Snapdragon 845 and Adreno 630 GPU. Unless stated otherwise, we report evaluation results on LG V50. RetinaFace [2] and MobileFaceNet [138] are implemented using TensorFlow 1.12.0 and converted to TensorFlow-Lite for mobile deployment. Image processing functions (edge detection, face cropping) are implemented using OpenCV Android SDK 3.4.3. The mobile device is connected to the server via a TCP connection.

Cloud. We implement the cloud side of **EagleEye** on a desktop PC running on Ubuntu 16.04 OS, equipped with Intel Core i7-8700 3.2 GHz CPU and an NVIDIA RTX 2080 Ti GPU (11 GB RAM). We implement most of the cloud-side functions in Python 3.5.2 and utilize Numba [141], a Just-In-Time (JIT) compiler for Python, to accelerate the performance comparable to C/C++. ICN and ResNet50-based ArcFace [21] are implemented using TensorFlow 1.12.0.



(a) Low.

(b) Medium.

(c) High.

Figure 4.16: In-the-wild dataset examples.

Table 4.3: Average and standard deviation of the composition of each face type in the test dataset.

	Low	Medium	High
Large frontal	3.00 ± 2.62	3.85 ± 2.11	5.20 ± 3.73
Large profile	1.00 ± 0.76	1.50 ± 1.49	2.8 ± 1.78
Low-resolution	3.07 ± 1.75	5.45 ± 2.50	8.87 ± 3.64
Total	7.07 ± 1.79	11.10 ± 3.74	16.87 ± 4.78

4.8 Evaluation

4.8.1 Experiment Setup

DNN training. We train our face detector on WIDER Face [142] train dataset. Also, we train our face recognizers (both the light and heavy models) on MS1M [124] dataset. ICN is trained on FFHQ dataset [143]. As FFHQ dataset does not contain face landmark labels, we employ state-of-the-art network [144] to estimate face landmarks and use them as ground truth labels.

Datasets. We evaluate **EagleEye** with two different datasets: *single faces* and *crowded scenes*. For single faces, we collect 50 identities in VGGFace2 [125] testset, with 50 samples per each identity. For the scenes, we use in-the-wild images (mostly containing faces of a single ethnicity group) collected and classified depending on crowdedness (i.e., *Low*, *Medium*, and *High*) as described in Chapter 4.3.1 (examples are shown in Figure 4.16). The detailed composition of

the faces in the scene dataset are summarized in Table 4.3. We also categorize the dataset depending on whether the target is present or not. Furthermore, we also collect scene images from WIDER Face [142] test dataset, which contains diverse ethnicity groups (15 images per each crowdedness category).

Evaluation protocols and metrics. We evaluate the performance of EagleEye with the following evaluation protocols and metrics:

- **Latency:** the time interval between the start and the end of the pipeline execution, measured on mobile.
- **Equal Error Rate (EER):** the value in the ROC curve where the false acceptance and false rejection rates are identical.
- **True Positive (TP) & False Positive (FP):** the rate in which the test faces are correctly/wrongly accepted as the target, respectively, given a fixed threshold.
- **Top-K accuracy:** the percentage of images in which the distance between the target face and the probe is within the top K-th among all faces in the scene (applies for scenes with the target present). This can also be interpreted as recall for a single target.
- **False alarm:** the percentage of images in which the system falsely detects that the target is present in the scene (applies for scenes with the target absent).

Comparison schemes. We compare the performance of EagleEye with the following comparison schemes:

- **2-step baseline** runs the conventional 2-step identification pipeline (composed of MobileNetV1-based RetinaFace and ResNet50-based ArcFace) all on the mobile sequentially.
- **3-step baseline** runs our proposed 3-step identification pipeline (composed of MobileNetV1-based RetinaFace, ICN, and ResNet50-based ArcFace) all on the mobile sequentially.

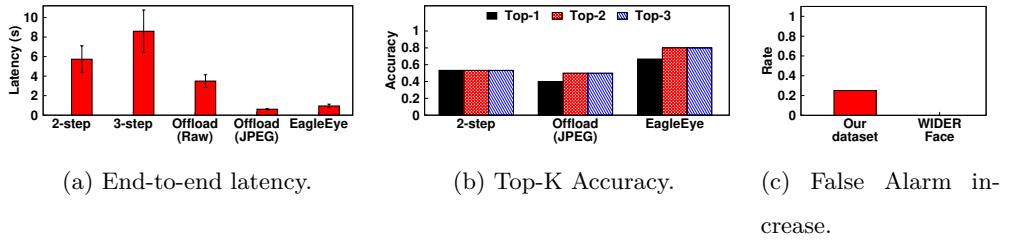


Figure 4.17: EagleEye performance overview.

- **Full offload** fully offloads the image to the cloud over LTE and runs the 3-step identification pipeline. The image is sent either raw or after JPEG compression. Note: we run this experiment under a normal LTE performance (≈ 11 Mbps), and it is likely that the performance of full offloading could be worse than what we report in crowded outdoor environments.

4.8.2 Performance Overview

We first evaluate the overall performance of **EagleEye** compared with alternatives for *High* scenes. Figure 4.17 shows the results. Firstly, as shown in Figure 4.17(a), **EagleEye** outperforms the latency of the 3-step baseline by $9.07 \times$ (with only 108 KBytes of data offloaded to the cloud). Also, it shows the highest Top-K accuracy (80% of Top-2 accuracy vs. 53% for the 2-step baseline) at the reasonable increase of false alarms (Figure 4.17(b) and (c)). A reason for the increase of the false alarm is that our dataset contains the faces of the same ethnicity group, increasing the chance of similar-looking identities with the target. For the WIDER Face dataset which contains more diverse ethnicity groups, we did not observe any false alarm increase. Note that the accuracy and false alarms are better with *Medium* and *Low* scenes, as shown in Figure 4.25.

Interestingly, while fully offloading JPEG-compressed images achieves the smallest latency, we observe that its Top-2 accuracy drops to 50% as shown in Figure 4.17(b), as compression artifacts hinder reconstruction performance of ICN and recognition network. We could apply video compression (e.g., H.264) to

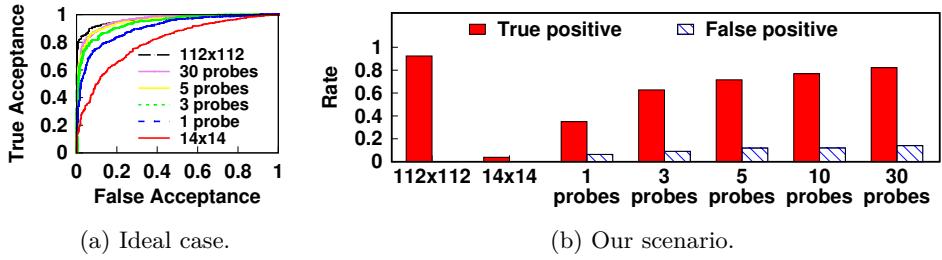


Figure 4.18: Performance of Identity Clarification Network.

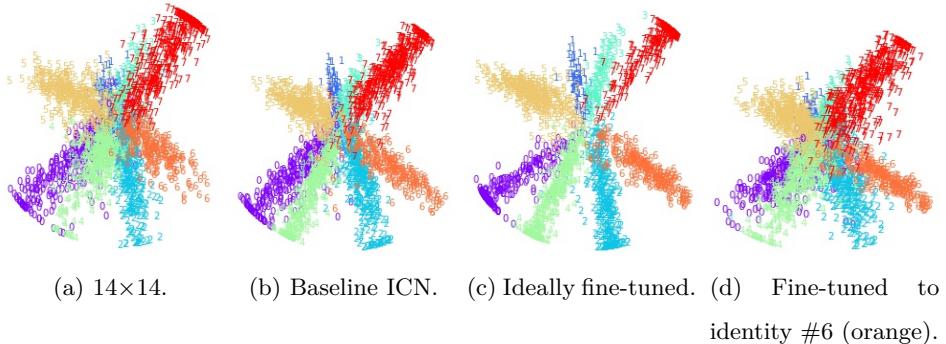


Figure 4.19: Feature map visualization for ICN.

minimize latency more, but it would further degrade performance as it adopts motion vector-based inter-frame encoding, incurring additional distortion in the faces. As compression artifact reduction is a challenging problem, recent attempts have been made to design specialized DNNs for it [145, 146]. Thus, we conjecture that solving this issue will not be trivial and leave detailed investigation as future work.

4.8.3 Identity Clarification Network

We evaluate the performance of ICN with a varying number of probes used for Identity-Specific Fine-Tuning. Figure 4.18 shows the results for (a) ideal cases (ICN trained for individual faces) and (b) our scenarios (ICN trained with a target identity), respectively. For the ideal case, ICN recovers the accuracy of 14×14 faces similar to 112×112 with about 5 probes only. For our scenarios,



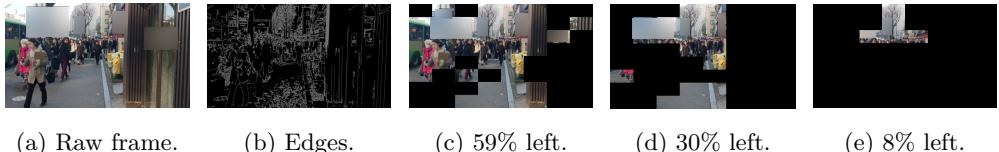
(a) 112×112 . (b) 14×14 . (c) Baseline. (d) Fine-tuned

Figure 4.20: Reconstruction example of ICN.

Figure 4.21: Background filtering.

as the number of probes increases, ICN injects more facial details of the target to the input LR face, significantly increasing the chance to identify the target with a relatively small increase in the FP. Figure 4.18(b) shows that the gain in TP (78%) outweighs that of FP (14%). We further analyze the reasons for accuracy improvement using a simple example with the 8 identities (the same setting as in Chapter 4.3.2). From the 14×14 LR faces whose features severely overlap with each other (Figure 4.19(a)), the baseline ICN (without fine-tuning) clusters each identity's features more tightly, but some overlapping regions still remain (Figure 4.19(b)). When enhancing each LR face with ICN fine-tuned with corresponding probes, we observe each feature cluster is separated even more clearly (Figure 4.19(c)). In the case of applying ICN fine-tuned to target identity #6 (orange samples), Figure 4.19(d) shows that the samples corresponding to the target are grouped to form a tight cluster. While other identity groups are pulled towards the target, the cases where the pulled samples overlap with those of the target (false positive) are not dominant.

Finally, Figure 4.20 shows the face reconstruction examples of ICN. Baseline ICN reconstructs a face quite similar to the ground truth but lacks some fine attributes (e.g., wrinkles) in the ground truth face. Identity-Specific Fine-tuning enables the ICN to instill such details in the reconstructed face, thus enabling accurate recognition.



(a) Raw frame. (b) Edges. (c) 59% left. (d) 30% left. (e) 8% left.

Figure 4.22: Example operation of Edge-Based Background Filtering.

4.8.4 Content-Adaptive Parallel Execution

4.8.4.1 Edge-Based Background Filtering

Next, we evaluate the performance of our Edge-Based Background Filtering method. Figure 4.21 shows the detection rate and latency gain as we increase the edge intensity threshold. Higher threshold results in higher latency gain, but at the cost of loss in detection rate. We observe threshold between 0.05 and 0.08 balances the tradeoff, and we empirically set it as 0.08 which achieves $1.76\times$ latency gain with 8.7% loss in detection rate. Figure 4.22 shows an example of image blocks being filtered for different thresholds (covered in black in Figure 4.22(c)–(e)). With a higher threshold, blocks containing large faces starts to get ruled out. The tradeoff can be more aggressively made if our system can only focus on identifying distant, small faces while relying on users to recognize large, closer faces.

4.8.4.2 Variation-Adaptive Face Recognition

To evaluate the effectiveness of Variation-Adaptive Face Recognition, we synthesize a group of faces, which contains 10 samples per each case classified in Figure 4.14. We compare our technique (adapting the recognition pipeline based on pose and resolution) with the following baselines: (i) running a lightweight recognizer (MobileFaceNet [138]) on all faces (denoted as *Base light*), (ii) running ICN and a heavy recognizer (ResNet50-based ArcFace [21]) on all faces (denoted as *Base full*), (iii) adaptively applying the lightweight and heavy rec-

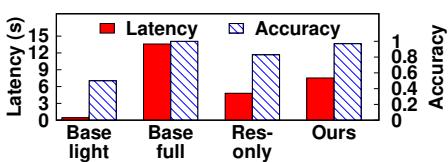
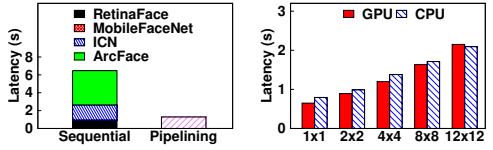
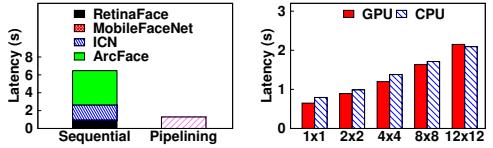


Figure 4.23: Performance of Variation-Adaptive Face Recognition.



(a) End-to-end latency.

Figure 4.24: Spatial Pipelining performance.



ognizers based on the resolution only (denoted as *Res-only*). We did not apply our parallel and pipelined execution for this experiment so that only the relative comparisons are meaningful.

Figure 4.23 shows that our approach achieves comparable accuracy with *Base full*, while reducing the latency by $1.80\times$. On the contrary, *Base light* and *Base full* suffer from low accuracy and significantly high latency, respectively. The *Res-only* yields fairly high accuracy gain with small latency overhead, but the accuracy remains lower than *Base full* as large profile faces processed by light MobileFaceNet results in inaccurate decisions.

4.8.4.3 Spatial Pipelining

Figure 4.24(a) shows the performance of Spatial Pipelining on *High* scenes. Our pipelining yields $5.03\times$ acceleration compared to the baseline that runs face detection and processes faces with Variation-Adaptive Face Recognition sequentially using the mobile GPU (denoted as *Sequential*).

We further analyze the effect of the number of blocks to parallelize. Figure 4.24(b) shows the latency of face detector with varying number of blocks. We need to divide the image in an overlapping manner to prevent faces split across blocks, which increases computational overhead due to repetitive face detection on the overlapping regions. Thus, the larger the number of blocks, the higher the latency overhead. Considering the tradeoff between such cost

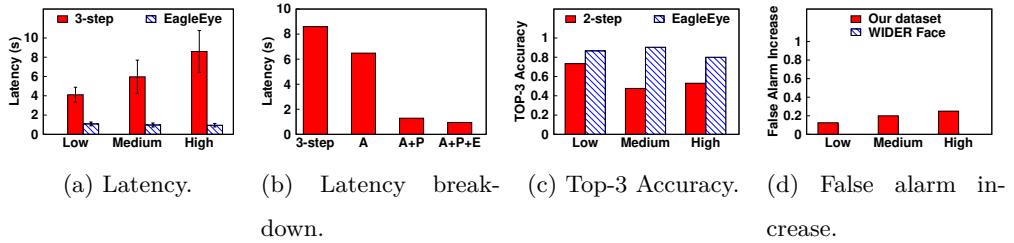


Figure 4.25: End-to-end latency for varying crowdedness.

and gain for parallelism, we divide the image into 4×4 blocks by default.

4.8.5 Performance for Varying Crowdedness

Figure 4.25(a) shows the end-to-end latency comparison of 3-step baseline and **EagleEye**. The latency of **EagleEye** remains similar regardless of crowdedness, mainly because we pipeline and parallelize the execution on mobile and cloud. However, the latency of 3-step increases with more crowded scenes since recognition latency increases proportionally to the number of faces. Accordingly, we conjecture that the latency gain will be greater as crowdedness increases even more. Furthermore, current bottleneck remains at the face detection stage, and we expect that the latency will be further reduced as face detectors become more optimized.

Figure 4.25(b) shows the latency breakdown on *High* scenes for gradually adding on the components of **EagleEye**: Variation-Adaptive Face Recognition (A), Spatial Pipelining (P), and Edge-Based Background Filtering (E). Combining each component yields a synergetic gain, achieving $9.07 \times$ acceleration compared to the 3-step baseline.

Finally, Figure 4.25(c) shows the Top-3 accuracy and false alarm increase of **EagleEye** compared to the 2-step baseline. Overall, **EagleEye** yields 27.6% accuracy gain, with accuracy above 80% even for *High* scenes. Figure 4.25(d) shows that at the cost of such accuracy gain, **EagleEye** results in 19.1% increased false alarm. Such increase is due mainly to the fact that our dataset contains

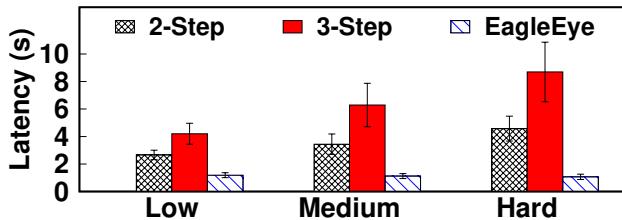


Figure 4.26: Latency evaluation on Google Pixel 3 XL.

the people with the same ethnicity, and we observe no increase in false alarm in case of WIDER Face dataset.

4.8.6 Performance on Other Mobile Devices

Lastly, we evaluate the end-to-end latency on Google Pixel 3 XL to validate the performance of **EagleEye** on other mobile devices. The inference times of MobileNetV1-based RetinaFace, ICN, ResNet50-based ArcFace, and MobileFaceNet are 918, 225, 193, 18 ms, respectively. Figure 4.26 shows that the latency performance of **EagleEye** and gain compared to 3-step baseline are similar ($8.14\times$ for *Hard* scenes) to previous results, indicating that **EagleEye** shows consistent performance on other devices.

Chapter 5

Pendulum: Network-Compute Joint Scheduling for Efficient and Accurate Live Video Analytics

5.1 Introduction

In this Chapter, we design **Pendulum**, an end-to-end system for fast and accurate cloud offloading-based live video analytics. For instance, a police agency deploys CCTVs and officers with AR glasses in a city for monitoring (e.g., criminal chasing, action recognition). Figure 5.1 illustrates such a deployment model, where multiple clients stream video over shared cellular radio access networks (RANs) to a GPU-equipped cloudlet located near the base station [147].

Live video analytics pipeline is composed of two stages: (1) *Network*: video streaming over network and (2) *Compute*: real-time Deep Neural Network (DNN) inference on edge server. The key to achieving high accuracy and throughput is to flexibly adjust the pipeline configuration (config) of video bitrate and DNN based on dynamic workload and resource availability, which is influenced by scene complexity [5, 10, 40], network bandwidth [85], and server contention [14]. However, we observe that achieving both goals is challenging, as resource bottlenecks alternate in complex patterns across the network and the compute stages.

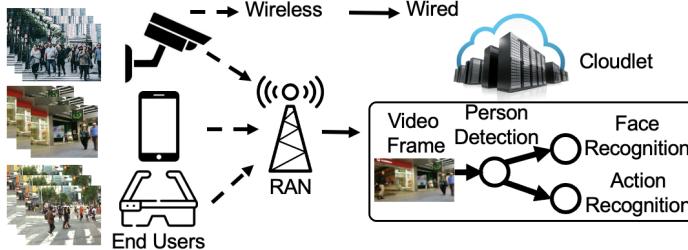


Figure 5.1: Scenario: cloudlet-based person monitoring.

In a person monitoring app in Chapter 2.2.3, network and compute bottlenecks occur 3 and 2 times each in a non-overlapping and switching pattern within a 30s window, resulting from latency fluctuations of video streaming (affected by encoding bitrate and channel status) and DNN inference (changed by the number of people in the scene).

While prior works support adaptation to fluctuating resources, they are limited to *single-stage scheduling* (i.e., network [1, 78, 85, 87] or compute [14, 40]).¹ They aim to minimize the target resource usage with minimal accuracy drop, assuming the other resource is sufficiently provisioned. However, they hardly handle alternating resource bottlenecks, and thus suffer from throughput/accuracy drop when the non-targeted resource is bottlenecked (Chapter 5.2.1). Also, a simple combination of network- and compute-only schedulers incur sub-optimal throughput/accuracy and resource cost (Chapter 5.2.2).

We present **Pendulum**, an end-to-end system designed for efficient and accurate live video analytics, employing a unique *network-compute joint scheduling* approach. Upon encountering a bottleneck in one stage, it conserves the resources in that stage and compensates for any potential drop in inference accuracy by utilizing excess resources from the other stage. Our approach offers two distinct benefits. First, it broadens the adaptation space. For example, by reducing the minimum bitrate that satisfies the app accuracy requirement (fa-

¹We refer to the single-stage scheduling as *controlling the network or compute resources independently*, albeit it indirectly affects the remaining stage (e.g., adapting video resolution changes DNN inference latency).

cilitated by using heavier DNNs for accuracy compensation), we can ensure a more reliable and higher throughput/accuracy than single-stage scheduling under bandwidth fluctuation. Second, it promotes a balanced allocation of network and compute resources, circumventing the need for resource over-provisioning.

Our joint scheduling strategy draws from insights into the interplay between video bitrate and DNN complexity. Complex DNNs, characterized by more layers and filters, can offset accuracy losses from lower bitrates by extracting more detailed visual features from scenes [148]. In particular, we found that a DNN with a large number of layers enables a contextual understanding of the scene (e.g., detecting an object not just by looking at it but also the objects around it), which improves accuracy in low bitrate videos (Chapter 5.3.3). This concept is supported by wider research in machine learning, highlighting that DNNs with more layers offer larger receptive fields for better scene interpretation [149, 150] and aligns with EfficientNet [148]’s principles of scaling input resolution, layers, and filters in tandem. Utilizing these insights, we navigate through a broader range of (bitrate, DNN) configs to satisfy application-specified accuracy requirements, thus adeptly handling alternating resource bottleneck scenarios.

While our approach sounds promising, realizing it entails the following challenges (Chapter 5.4.1). (i) *Overhead for resource demand profiling*. (Bitrate, DNN) configs meeting the app accuracy requirement vary with the video content, yet profiling them requires executing multiple DNN inferences across different encoding bitrates, leading to substantial latency costs (e.g., 3.4s). (ii) *Inaccuracy in resource availability monitoring*. Precisely estimating resource availability proves difficult, especially for RAN bandwidth. Conventional methods based on packet arrival statistics for video analytics [85, 151] fall short of accurately assessing the RAN’s available excess bandwidth. (iii) *Complexity in resource scheduling*. It is challenging to allocate shared bandwidth and GPU resources to multiple users, each with its own set of resource demands and chan-

nel conditions. The scheduling falls into a multi-dimensional knapsack problem (NP-hard) with a large search space caused by 2D (bitrate, DNN) config space.

We address the challenges with the following ideas (Chapter 5.4.2).

- *Video content-aware demand profiler* that activates with significant content changes, profiling minimal set of configs and using interpolation to estimate others based on network and compute dependencies (Chapter 5.5.1).
- *RAN-aware availability monitor* that utilizes the RAN Intelligent Controller (RIC) in O-RAN architecture to gather real-time Resource Block scheduling information, translating it into application layer bandwidth availability (Chapter 5.5.2).
- *Network-compute joint resource scheduler* that implements our *iterative max-cost gradient* algorithm to find approximate solutions for joint resource scheduling. This algorithm optimizes user-specific configs based on the maximum cost gradient, aiming to alleviate bottlenecks (Chapter 5.6).

We implement **Pendulum** on OpenAirInterface 5G RAN software stack [152, 153] and multi-GPU edge server, with app-level C++ scheduler. We conduct an extensive evaluation with various video datasets and state-of-the-art (SOTA) DNNs; **Pendulum** achieves up to 0.64 mIoU (mean Intersection-over-Union) accuracy gain (from 0.17 to 0.81) and 1.29 \times higher throughput compared to SOTA single-stage scheduling systems. **Pendulum** also achieves 25% lower resource cost than the network-compute decoupled scheduling baseline.

Our key contributions are summarized as follows:

- To our knowledge, **Pendulum** is the first live video analytics system with network-compute joint scheduling.
 - We design an end-to-end system for joint scheduling, composed of (i) efficient and scalable joint scheduling mechanism and (ii) RAN-aware multi-user joint resource scheduling algorithm.
 - We conduct extensive evaluation on various datasets and state-of-the-art

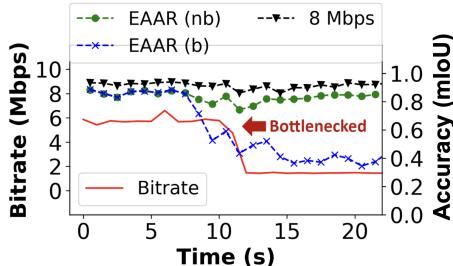


Figure 5.2: Performance analysis of network-only scheduling with EAAR [1] (b: bottleneck, nb: no bottleneck).

DNNs. **Pendulum** achieves up to 0.64 mIoU gain (from 0.17 to 0.81) and 1.29 \times higher throughput compared to state-of-the-art single-stage scheduling systems. **Pendulum** also achieves 25% lower cost than decoupled baseline in multi-user scheduling.

5.2 Limitations of Prior Works

To motivate **Pendulum**, we analyze why prior single-stage scheduling systems fail in alternating resource bottleneck scenarios analyzed in Chapter 2.2.3.

5.2.1 Limitations of Single-Stage Scheduling

Throughput/accuracy drop. It is challenging to simultaneously achieve high throughput and accuracy by controlling only a single stage. Figure 5.2 shows the changes in bitrate and accuracy over time when serving a MOT17 [34]-04 video over an emulated RAN using EAAR [1], a network-only scheduling system that optimizes bitrate by adjusting the encoding quality of regions of a frame depending on whether or not objects were present in the previous frame. As an accuracy metric, we use the mean intersection over union (mIoU), which computes the overlap between the ground truth bounding box and the inference output. EAAR effectively adjusts the bitrate to \approx 5.68 Mbps with minimal accuracy drop compared to 8 Mbps encoding (EAAR (nb)). However, the through-

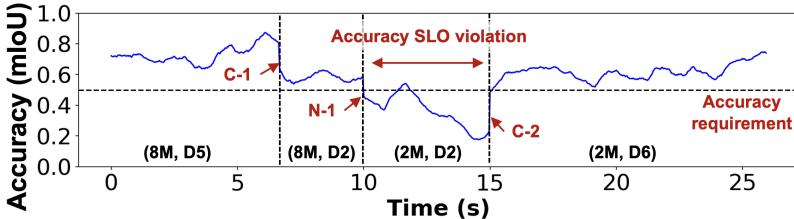


Figure 5.3: Accuracy changes while running decoupled schedulers. C-1/2 and N-1 denote compute and network scheduling events, respectively.

put and accuracy drop when the network bandwidth drops below its capability. When we throttle the bandwidth to 2 Mbps at $t=10s$ using `tc` [154], EAAR reduces the bitrate accordingly, incurring a significant mIoU drop (EAAR (b)).

Resource waste from over-provisioning. Allocating the right amount of resources to the non-targeted stage (e.g., compute stage in the case of EAAR [1]) is also challenging to avoid bottlenecks or wastage due to under/over-provisioning. For example, in Figure 2.2(b), allocating 1 GPU to the user results in a 14% latency violation rate, while allocating 2 GPUs incurs 62% resource waste. Elastic resource provisioning has higher operational costs [155] or may not be possible for edge server scenarios (e.g., dedicated private cluster [156]).

5.2.2 Why Simple Combination of Two Schedulers Fails?

Simply running two network and compute-only schedulers in a decoupled manner (where each scheduler is unaware of the other) is also limited for two reasons. First, uncoordinated scheduling timing leads to drops in throughput/accuracy. For example, Figure 5.3 shows accuracy changes when serving the BDD [157] video initially encoded at 8 Mbps with the EfficientDet [108] object detector that has D0–D6 candidate backbones with varying complexities by using decoupled bitrate (EAAR [1]) and DNN adaptation (Chameleon [40]) schedulers. Suppose the DNN scheduler first reduces the DNN from D5 to D2 (according to accuracy profiling results) to optimize resource usage without violating the ac-

curacy requirement (C-1, $t=6$ s). At $t=10$ s, the network bandwidth is throttled to 2 Mbps. The bitrate scheduler reduces the bitrate to below the bandwidth to avoid throughput drop (N-1), violating the accuracy requirement. The problem persists until the next scheduling interval of the computer scheduler (C-2, $t=15$ s) when it increases the DNN back to D6. The problem may be alleviated with more frequent scheduling, but it incurs overhead, especially for accuracy profiling, often done by running multiple candidate DNNs [40].

In addition, decoupled scheduling leads to sub-optimal resource allocation and waste in multi-user scenarios. This is mainly because it is unaware of different users' sensitivities to the network-compute demand curve that depends on video contents (i.e., a certain user may require more resources on the remaining stage for compensation when reducing bottleneck stage resources). For example, equally reducing resource usage across multiple users without such knowledge incurs 25% additional resource cost (Chapter 5.7.5).

5.3 Our Approach

5.3.1 Goals

Our goal is to design an end-to-end scheduling system that effectively handles alternating resource bottlenecks while achieving the following properties.

High throughput and accuracy. We aim at end-to-end scheduling across end users and edge servers to simultaneously achieve high throughput (e.g., real-time 30 fps processing) and high DNN inference accuracy.

Minimal operational costs. We also aim to minimize operational costs. Specifically, network and compute costs vary depending on operators (e.g., 1 Mbps streaming over 5G networks: \$0.36 [158]–\$0.54 [159] per hour, V100 GPU on cloud services: \$0.74 [160]–\$0.91 [161] per hour). Our goal is to efficiently provision network/compute resources to avoid wastage from over-provisioning.

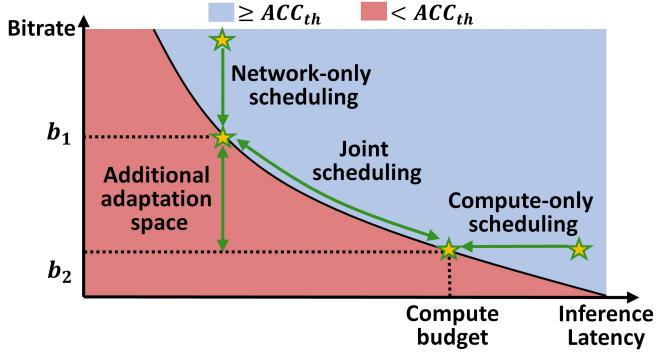


Figure 5.4: Single-stage vs. joint scheduling comparison.

No modification on the RAN scheduler. For generality, we impose no modifications on the RAN scheduler, which is proprietary to the network operator.

5.3.2 Key Idea: Joint Scheduling

Our approach is to *jointly* schedule the network and compute resources. 5.4 shows a comparison between single-stage and joint scheduling. Network-only scheduling, which controls the video bitrate with a fixed GPU utilization, can only reduce the bitrate up to b_1 without violating the accuracy requirement (ACC_{th}). If network bandwidth drops below b_1 , network bottleneck occurs, dropping the throughput. However, joint scheduling utilizes additional compute resources (up to the compute budget) to further reduce the bitrate to as low as b_2 . This approach is feasible as, although the network and compute bottlenecks alternate over time, they occur independently, and thus, surplus resources are mostly available on the non-bottleneck stage. For example, in 2.2, the Pearson correlation coefficient of the two bottleneck events is -0.04, indicating a weak correlation.

Figure 5.5 shows an example scenario that demonstrates the benefits of joint scheduling. When the network bandwidth is sufficient, the joint scheduler uses a high bitrate (e.g., 5 Mbps) and runs a lightweight EfficientDet-D0 [108]

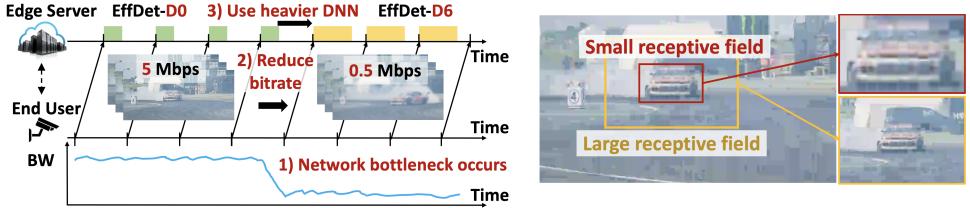


Figure 5.5: Joint scheduling example for network bottleneck scenario.

Figure 5.6: Illustration of the impact of the receptive field.

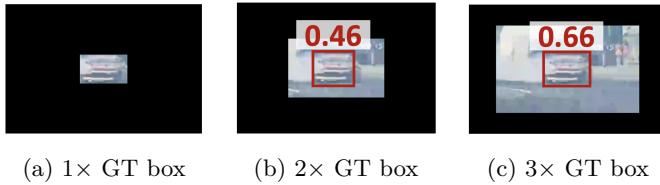


Figure 5.7: Example detection results (box and confidence) for different crop sizes around ground truth (GT) box.

object detector. When the bandwidth drops, the bitrate is reduced accordingly to 0.5 Mbps so that the video can be streamed in real-time. At the same time, it uses additional compute resources by running the heavier EfficientDet-D6 to compensate for the possible loss of accuracy due to reduced video quality. Similarly, it can reduce the DNN complexity and increase the bitrate if the compute becomes a bottleneck.

Note that the joint scheduling is orthogonal to optimization techniques in existing single-stage scheduling systems (e.g., ROI encoding [1] and frame filtering [78]) and can be generally integrated into many state-of-the-art systems. We show its benefits on DDS [87] and EAAR [1] in Chapter 5.7.2.

5.3.3 Why is Joint Scheduling Possible?

We analyze why a heavier DNN can compensate for the accuracy drop due to a low bitrate (and vice versa). Heavy DNNs with many layers and filters can capture diverse complex features [148]. Especially, their large receptive field

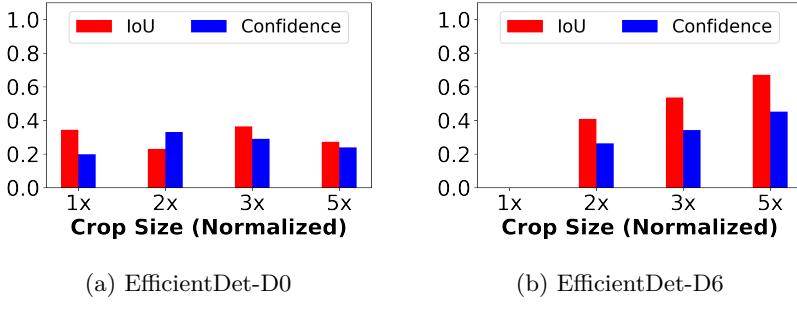
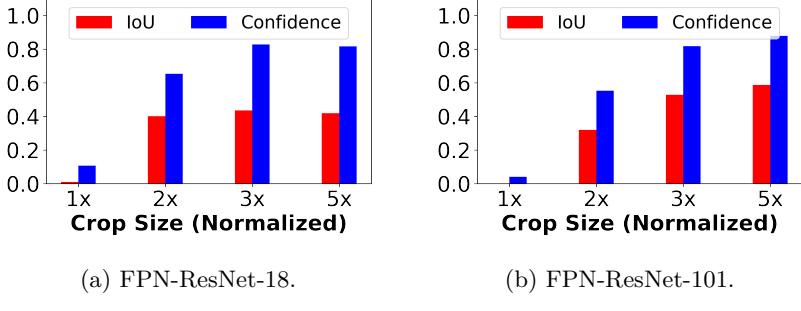


Figure 5.8: Detection accuracy in low-bitrate video.

size (i.e., how large area a DNN analyzes to detect objects) [149], which is proportional to the number of layers [150], helps achieve high accuracy in low-bitrate videos by analyzing the scene context with a wider view. For example, as shown in 5.6, for low-bitrate videos, it is challenging to recognize the blurry object as a car just by looking at it, whereas a large receptive field allows for analyzing the surrounding context (e.g., road lane, nearby car) for accurate recognition.

We quantitatively verify this on two common tasks: object detection and segmentation. We emulate the effect of the receptive field by restricting the input context information to a DNN; this is done by masking the DAVIS17 [162] video (containing a single dominant object) video frame around the ground truth object bounding box as shown in Figure 5.7. We run EfficientDet [108] with D6 and D6 backbones consisting of 49 and 134 layers on the masked frame and measure the accuracy (IoU of the object). Figure 5.8(a) shows that D0’s accuracy remains low, regardless of the crop size. However, Figure 5.8(b) shows that D6’s accuracy constantly improves with larger crop size, indicating that it leverages its large receptive field for accurate detection. We observe a similar trend for segmentation task: Figure 5.9 shows the accuracy for the same video using FPN [163] with ResNet-18 and ResNet-101 backbones.



(a) FPN-ResNet-18. (b) FPN-ResNet-101.

Figure 5.9: Segmentation accuracy in low-bitrate video.

5.3.4 Joint Scheduling Problem Formulation

Our joint scheduling aims to allocate network and compute resources among users while satisfying resource constraints and minimizing total resource cost, given the following as input: Each user i has K_i accuracy-satisfying (bitrate, DNN) configurations, $C_{i,j} = (b_{i,j}, t_{i,j})$, where $j \in [1, K_i]$ and $b_{i,j}$ and $t_{i,j}$ are the candidate encoding bitrates and inference latencies of the candidate DNNs, respectively. The user streams video at f_i fps, performs n_i inferences per frame, and experiences spectral efficiency R_i (Mbps per Resource Block (RB)). There are $N_{RB,total}$ RBs per each scheduling window, and the server has N_{GPU} GPUs, with available utilization time t_{th} (s). Given this input, our joint scheduler finds the cost-minimizing allocation $J^* = \{j_1^*, \dots, j_N^*\}$ by solving:

$$\begin{aligned} \min_J \quad & Cost_{Network}(\sum_i b_{i,j}) + Cost_{Compute}(\sum_i t_{i,j}) \\ \text{s.t.} \quad & \sum_i f_i \cdot n_i \cdot t_{i,j} \leq t_{th} \cdot N_{GPU}, \\ & \sum_i N_{RB,i,j} \leq N_{RB,total}, \text{ where } N_{RB,i,j} = b_{i,j}/R_i \quad \forall i \end{aligned} \quad (5.1)$$

The first constraint enforces that the total inference latencies do not exceed the compute budget. The second constraint enforces that the allocated bitrates converted to RBs do not exceed the available RBs in the RAN.

5.4 Design Overview

5.4.1 Challenges

While our joint scheduling approach sounds promising, realizing it in practical settings presents three challenges:

Resource demand profiling overhead. Estimating which (Bitrate, DNN) configs satisfy the app accuracy requirement is challenging due to two reasons (Chapter 5.5.1). First, the demand dynamically changes depending on video content (e.g., object speed, lighting condition). Second, due to black-box nature of DNN inference, it is inevitable to profile them by running multiple candidate DNNs over the video encoded into multiple bitrates, incurring significant overhead (e.g., full search of 5 bitrates \times 7 DNNs takes 3.4s on an RTX 2080 Ti GPU).

Inaccuracy in monitoring resource availability. Accurately tracking resource budgets is also non-trivial. Especially, network bandwidth estimation techniques in conventional packet arrival statistics-based video streaming/analytics cannot be applied in joint scheduling (Chapter 5.5.2), as they cannot estimate the RAN’s surplus bandwidth capacity determined by idle number of RBs and users’ spectral efficiencies (Chapter 5.5.2).

Complexity in resource scheduling. Allocating shared RAN and server resources to multiple users is challenging. This is because users have different resource demand sensitivities (i.e., amount of resource required to compensate when reducing the bottleneck stage resource by a unit). Moreover, due to user’s different channel status, RAN’s total bandwidth capacity heavily varies depending on scheduling decisions (e.g., by up to 9 \times) [164], which is not considered in prior multi-user video analytics scheduling (e.g., in GPU clusters [5,93] or shared wired network [156]). Such scheduling falls into a multi-dimensional knapsack problem (NP-hard) with a large search space due to the (bitrate, DNN) config

space.

5.4.2 Key Ideas

We address the above challenges with three key ideas.

Video Content-aware Demand Profiling (Chapter 5.5.1): We design a run-time profiler that estimates resource demands with minimal overhead. It triggers profiling only upon significant content change. During each profiling event, it only profiles the accuracy of a minimum number of configs and models the dependencies of the network and compute stages to interpolate the rest (e.g., accuracy gain from increasing bitrate may saturate as DNN becomes heavier).

RAN-aware Availability Monitoring (Chapter 5.5.2): We leverage the RAN Intelligent Controller (RIC) in the standard O-RAN architecture to obtain real-time physical layer Resource Block (RB) scheduling information from the RAN and convert it to app layer bandwidths.

Demand Curve-aware Scheduling (Chapter 5.6): To make the NP-hard scheduling problem tractable, we devise an *iterative max-cost gradient algorithm* that computes an approximate solution with $O(MN)$ complexity for N users with M configs. It first finds the user-wise optimal configs, and iteratively adjusts the user with the *maximum cost gradient* (i.e., maximum expected decrease in bottleneck resource usage by increasing a unit usage of the other resource) until bottleneck resolves.

5.4.3 System Architecture

Figure 5.10 shows the architecture of Pendulum and its end-to-end workflow.

Data Flow. Users specify their app QoS requirements (latency, accuracy) per User Equipment (UE), and each UE (we interchangeably use UE and user throughout the paper) streams their live video encoded at the target bitrate.

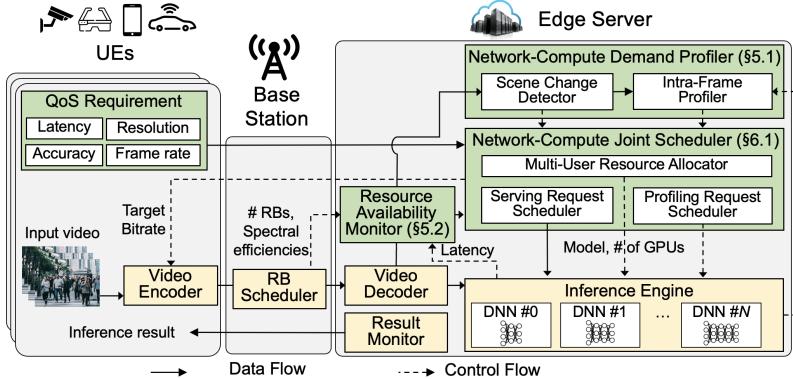


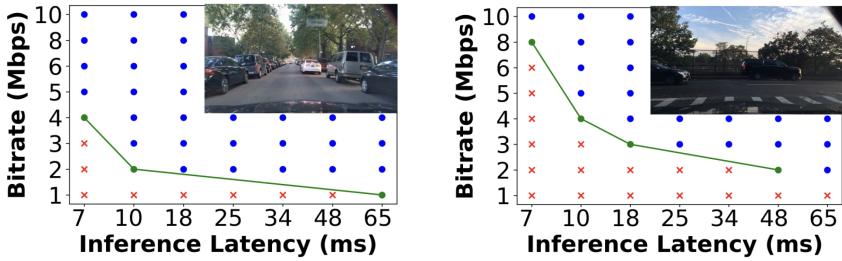
Figure 5.10: Pendulum system architecture (Yellow: video analytics pipeline, Green: Pendulum components).

After receiving and decoding the frames, the server performs the target DNN inference and aggregates the results.

Control Flow. *Network-compute joint scheduler* finds resource allocations across users to minimize overall resource usage (Chapter 5.6). To provide the resource demands and availability needed for scheduling, the *network-compute demand profiler* (1) dynamically triggers profiling only during large scene changes and (2) minimizes the number of config lookups (Chapter 5.5.1). Profiling results are combined with resource usage and budget monitored by the *resource availability monitor* (Chapter 5.5.2) for joint scheduling enabled by controlling video bitrate and DNN complexity knob pair (Chapter 5.5.3).

5.5 Joint Scheduling Mechanism

We describe key components of Pendulum that efficiently profile resource demands, monitor resource availability, and apply joint scheduling decisions.



(a) Static scene, good lighting

(b) Dynamic scene, poor lighting

Figure 5.11: Demand curves for different scenes. Blue/red points: configs above/below the accuracy requirement, green curve: Pareto-optimal configs.

5.5.1 Network-Compute Demand Profiler

5.5.1.1 Goal and Challenges

The profiler's goal is to analyze the Pareto-optimal demand configs of the user's live video stream. We define that a (bitrate, DNN) config (b, t) is Pareto-optimal if (b, t) satisfies the accuracy constraint and no other (b', t) and (b, t') exists s.t. $b' < b$, $t' < t$. Profiling is challenging due to two reasons.

- Fast-changing demand curve:** Pareto-optimal configs frequently change over time depending on the scene content. Figure 5.11 shows an example on a BDD [157] video for 8 bitrates and 7 EfficientDet [108] backbones. In a static scene (the car is not moving) with good lighting conditions (Figure 5.11(a)), it is easy to detect objects from a low-bitrate video with lightweight backbones. Thus, the Pareto-optimal configs have small b, t values. In contrast, the values become larger for dynamic scenes (the car is moving) with poor lighting conditions (Figure 5.11(b)). Changes can be fast; cost-optimal YOLOv5 [33] backbone for 4 Mbps bitrate changes every 4.4s for BDD [157] video.
 - High profiling overhead:** Each demand profiling event requires a 2D (bitrate, DNN) search space, involving repetitive DNN inferences over a frame encoded in multiple bitrates. Frequent profiling (e.g., fixed-interval, periodic [40]) incurs significant resource overhead. For example, exploring all

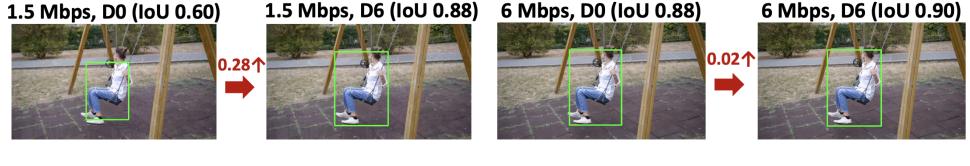
search space composed of 7 EfficientDet backbones (D0-D6) and 5 bitrates over a single frame takes 3.4s on RTX 2080 Ti GPU. Triggering this every 2s incurs \approx 50% overhead compared to 30 fps inference serving).

5.5.1.2 Lightweight Scene Change Detector

Two categories of features can be utilized for scene change detection: heavy-but-accurate high-level (e.g., SIFT [165]), SURF [166]) and lightweight-but-noisy low-level. We adopt a lightweight feature ensemble approach for fast and accurate scene change detection. We choose the following features, which capture different aspects of scene changes in a complementary way. They are highly correlated with the demand changes, allowing for effective skipping of unnecessary profiling (Chapter 5.7.5).

- **Camera motion:** We employ the Average Motion Vector [1] to identify scene transitions. A scene change is detected when the sum of the average motion vector magnitudes (derived directly from the video codec, thus eliminating additional processing) exceeds the threshold th_1 .
- **Object motion:** A scene change is also gauged by observing the bounding box drift. We assert a change when the mIoU metric between the reference frames and the current frames drops below the threshold th_2 .
- **Lighting condition:** A scene change is ascertained when the chi-square distance between the histograms of the reference and the current frames, called Color Histogram Difference [18], exceeds the threshold th_3 .

We detect scene change when two or more conditions are met. We use the MOT [34] and BDD [157] datasets to empirically set the thresholds th_1 , th_2 , and th_3 as 0.5.



(a) 1.5 Mbps bitrate.

(b) 6 Mbps bitrate.

Figure 5.12: Accuracy gain from DNN backbone increase (D0 to D6) varies depending on video bitrate.

5.5.1.3 Intra-Frame Profiler

To avoid an exhaustive full search of multi-dimensional config space, we propose the *weighted multi-knob accuracy interpolation* inspired by the linear multi-knob accuracy interpolation introduced by Chameleon [40]. The prior approach is based on the knob independence assumption: accuracy gain from increasing one config value is independent of the remaining configs. Thus, it profiles each config axis (e.g., bitrate, DNN) with fixed remaining-stage values and interpolates the rest. However, such an approach misses the interaction effect between the network and the compute stages, thus resulting in low profiling performance (e.g., low accuracy or low config search reduction). We improve this by adopting *weights*. Figure 5.12 illustrates our key insight: IoU increase when changing the backbone from D0 to D6 saturates as the bitrate is higher (as for high-quality 6 Mbps bitrate the D0 backbone already finds the object accurately). To take this into account, weighted multi-knob interpolation works as follows. Suppose it profiles the backbone axis with bitrate fixed to b_0 . Then it interpolates the accuracy gain from increasing the DNN backbone from D_j to $D_{j+\Delta}$ with bitrate b_i as

$$ACC(b_i, D_{j+\Delta}) - ACC(b_i, D_j) = w(b_i - b_0) \times (ACC(b_0, D_{j+\Delta}) - ACC(b_0, D_j)) \quad \text{for } 0 \leq i \leq N. \quad (5.2)$$

$ACC(b_i, D_j)$ is the accuracy of the config (b_i, D_j) , and $w(b_i - b_0)$ is the weight factor; for Chameleon [40], $w(b_i - b_0) = 1$. We model $w(b_i - b_0)$ as linear, and

fit $w(0) = 1$ and $w(b_N - b_0) = 0.3$ using our evaluation datasets (Chapter 5.7). For ground truth labels, we use golden config output (i.e., heaviest backbone on highest bitrate) similar to prior works [40, 78, 85].

5.5.1.4 DNN Inference Latency Profiler

The inference latencies of all candidate DNNs are profiled offline, stored as a look-up table, and updated online upon each inference completion. For two-stage tasks (composed of detection and analysis as in Equation 2.1), we also track the number of objects per frame with moving average filtering. To improve latency predictability of DNN inference, we make the following design decisions similar to prior work [167].

No multithreading. We do not use multithreading across users, as it only achieves a 25% throughput gain at the cost of $\approx 100 \times$ tail latency increase [167].

No model loading latency. Considering recent GPU memory sizes (e.g., 24 GB for Titan RTX, 48 GB for RTX A6000), we assume that all backbones are loaded into GPU memory (e.g., 6.5 GB for 7 EfficientDet [108] backbones, 10.4 GB for 7 FPN [168] backbones), and thus no model switching overhead.

5.5.2 Resource Availability Monitor

5.5.2.1 RAN-Informed Network Availability Monitor

Network availability is determined by the number of available RBs and the user’s channel status (per-RB bandwidth or spectral efficiency). Conventional video streaming/analytics systems (e.g., WebRTC) use packet size and arrival timestamps for bandwidth estimation [85, 151]. However, such approaches relying on packet-level information on the application side have a critical limitation in that they cannot estimate the *remaining* bandwidth capacity (i.e., how much additional bandwidth we can use when the compute becomes a bottleneck). Fig-

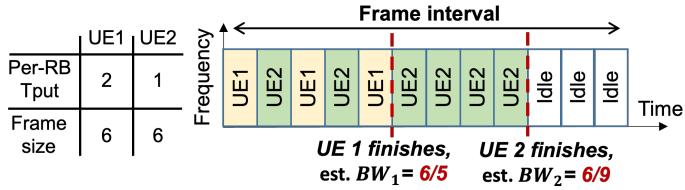


Figure 5.13: App-side packet-level bandwidth estimation cannot know RAN's remaining bandwidth capacity.

Figure 5.13 shows a 2-user example (frame sizes 6 and spectral efficiencies 2 and 1). Packet-level estimation not only underestimates spectral efficiency (due to sharing from underlying RB scheduler, e.g., Proportional Fair), but also cannot know how many idle RBs remain.

To address this problem, we use the RAN’s physical layer RB scheduling information for accurate bandwidth capacity estimation. While RAN-awareness has been proven to improve app performance (e.g., video streaming [169], web browsing [170], congestion control [171]), applying existing solutions is limited in two aspects. First, they often require modifications in cellular firmware (e.g., to decode control channel [171]), which is unavailable in commodity devices. Second, they are user-side solutions, incurring a monitoring delay when aggregated by the server for scheduling.

Instead, we use RAN Intelligent Controller (RIC) in the O-RAN standard [172] to obtain information from the base station (eNB/gNB) to the edge server. RIC has recently been optimized for <10 ms monitoring delay [173,174], which enables accurate resource estimation and scheduling. Specifically, we use E2 [175], a near-real time interface between the base station and app (xApp). E2 defines the functionality that can be monitored or controlled as service models (SMs), and we build our interface atop KPM (Key Performance Metric) SM [176] for resource monitoring. Figure 5.14 shows the RIC message format used in Pendulum:² we monitor the total number of RBs, the number of users

²Microsoft's recent Programmable RAN with dynamic SM [177,178] makes dynamic config

```

struct sched_info{
    int N_total_RBs;
    int n_users;
    ue_stats* stats;
};

struct ue_stats{
    double TBS;
    int N_RBs;
};

```

Figure 5.14: RIC message format.

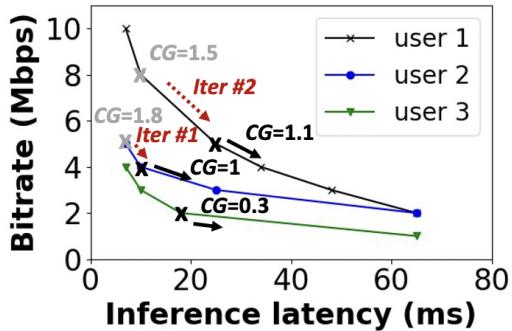


Figure 5.15: Iterative Max Cost Gradient algorithm example (2 iterations, CG : cost gradient).

and allocated RBs, and the achieved Transport Block Size (TBS) for each interval (e.g., 1s). Using this, we monitor UE i 's spectral efficiency by

$$R_i = (1 - OH) \cdot (TBS/N_{RB,i}) \cdot (1/(1 + \epsilon)), \quad (5.3)$$

where $N_{RB,i}$ is the number of allocated RBs and TBS is the PHY layer Transport Block Size. OH is the physical channel overhead (3GPP TS 38.306 [35] models it as 0.08 for UL in frequency range FR1). ϵ is the transport to physical layer protocol overhead: 0.068 for 5G networks [171].

5.5.2.2 Compute Availability Monitor

In our deployment model, there is a single, dedicated edge server for inference serving with fixed compute (GPU) availability, simplifying compute availability monitoring. However, GPU utilization must be monitored on the fly in future scenarios where other competing tasks exist.

5.5.3 Joint Scheduling Knob Controller

Video encoding bitrate can be modeled with three knobs
of RAN monitoring more flexible.

$$\text{Bitrate} \propto \text{Frame rate} \times \text{Resolution} \times \text{Pixel Quantization} \quad (5.4)$$

where several efficient knob control methods have been designed (e.g., frame rate [75, 78–82], resolution [40, 77, 179], pixel quantization [1], or a combination of all [85]). Similarly, DNN inference latency can be modeled as:

$$\text{Latency} \propto \text{Enhancement} + \text{Backbone} \times \text{Quantization} \quad (5.5)$$

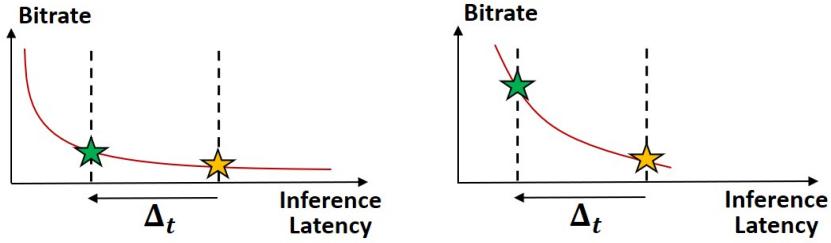
where Enhancement represents the input preprocessing step (e.g., super-resolution) to improve frame quality before inference, Quantization is applied to the DNN weights (e.g., float16, int8) to reduce computation, and Backbone refers to the feature extractor of a DNN, which is characterized by the DNN architecture (e.g., number of layers and channels).

While joint scheduling is not limited to specific knobs, we choose (pixel quantization, backbone) due to two reasons. First, pixel quantization controls the bitrate without affecting the compute stage (e.g., adapting frame rate or resolution unexpectedly changes DNN inference latency or the number of inference jobs). We control Quantization Parameter (QP) commonly exposed in video codecs (e.g., H.264/265). Second, DNNs for various tasks commonly support diverse backbones with wide accuracy-latency tradeoffs (e.g., ResNet-18/50/101/152 or EfficientNet-B0 to B7) [2, 33, 108, 148, 180]. We also show the effectiveness of joint scheduling with other knob choices in Chapter 5.7.3.

5.6 Joint Scheduling Algorithm

So far, we have described how **Pendulum** collects the necessary information for joint scheduling and how to apply a scheduling decision. We now describe how **Pendulum** effectively solves the scheduling problem in Chapter 5.3.4 for the given information.

Figure 5.16 motivates the importance of good joint scheduling. Suppose two users (with different Pareto-optimal resource demand curves) using the



(a) User A’s demand curve.

(b) User B’s demand curve.

Figure 5.16: Bandwidth required to compensate Δ_t inference latency differs depending on the demand curve.

configs marked as yellow stars. When a compute bottleneck occurs, the total inference latency should be reduced by Δ_t . Reducing user B’s backbone requires a significantly larger bitrate increase for accuracy compensation than adjusting user A’s, likely resulting in inefficient overall solutions. However, such scheduling is challenging due to the large search space; it involves exploring $O(M^N)$ options for N users (e.g., 10s-100s), each with a large number of M configs (e.g., 49 for 7×7 bitrate and backbones). This scheduling falls into a multi-dimensional knapsack problem, which is NP-hard.

To make the problem tractable, we design an *iterative max-cost gradient algorithm* that (1) finds user-wise cost-optimal configs (from the demand curves profiled by the demand profiler in Chapter 5.5.1) and (2) incrementally adjusts the allocation until bottleneck resolves. This heuristic efficiently finds near-optimal solutions in a reduced $O(MN)$ search space.

Algorithm 2 describes the algorithm. The scheduler takes the most recent profiled accuracy-satisfying demand curve $\{C_{i,j}\}$ for each user i along with the resource budgets $B_{network}$ and $B_{compute}$ as input, and outputs the cost-minimizing resource allocation S . First, it finds user-wise cost-optimal configs (lines 1–2). Then, it checks if the selected configs exceed the resource budgets. If a bottleneck is detected (line 2), it iteratively adjusts the config of the user with

Algorithm 2 Iterative Max Cost Gradient Algorithm

Inputs: Accuracy-satisfying configs $C_i = \{(b_{i,j}, t_{i,j})\}$ for users $1, \dots, N$, resource budgets $B_{network}, B_{compute}$.

Output: Cost-minimizing resource allocation $S = \{j_1, \dots, j_N\}$

- 1: $S \leftarrow \{0, \dots, 0\}$ i in $1, \dots, N$
 - 2: $S[i] \leftarrow GetCostOptimalConfig(C_i)$
 DetectBottleneck($S, B_{network}, B_{compute}$) == true
 - 3: $j \leftarrow FindMaxCostGradientUser(C, S)$
 - 4: $S[j] \leftarrow AdjustConfigByStep(C[j], S[j])$
 - 5: return S
-

maximum cost gradient by a step until the bottleneck is resolved (lines 3–4). The cost gradient is defined as how much the resource cost of the bottleneck stage can be reduced by increasing the resource cost of the non-bottleneck stage. For example, if the network becomes a bottleneck, user i (with currently selected config j)'s cost gradient CG_i is

$$CG_i = \left| \frac{Cost_{Network}(b_{i,j+1} - b_{i,j})}{Cost_{Compute}(t_{i,j+1} - t_{i,j})} \right|, \quad (5.6)$$

assuming $\{(b_{i,j}, t_{i,j})\}$ is sorted ascending order of t .

Figure 5.15 shows an example for three users. Suppose user-wise cost-optimal configs are black crosses. If the bitrate sum exceeds the network budget, the algorithm first reduces user 2's bitrate, which has the highest cost gradient (1.8) (*Iter #1*). Then, it selects user 1 who has the largest cost gradient (*Iter #2*). The process is repeated until the bottleneck is resolved.

Exception handling. Pendulum may not be able to find feasible solutions in two cases. The first is when a user has no accuracy-satisfying config under resource budget. In such a case, we reduce the bottlenecked resource (for real-time processing) and increase the usage of the remaining stages to maximum (for best-effort accuracy). The second is when both stages are bottlenecked (happens very rarely as analyzed in Chapter 2.2.3). In such a case, Pendulum reports the user (video analytics system operator) achievable accuracy under

current resources for further action (e.g., lower the accuracy requirement or secure more resources).

5.7 Evaluation

Implementation. We implement the demand profiler and joint resource scheduler in the application layer in C++. Network bandwidth estimator is implemented in C++ using FlexRIC [174] (O-RAN’s E2 [175]-compliant RIC SDK) for communication between base station and edge server. We currently run the base station and the DNN inference engine in the same server, and thus, the RIC communication delay is negligible, whereas, in practical MEC scenarios, we expect \approx 1-10 ms delay [4,174], which is still tolerable considering the scheduling interval (e.g., 2s). To stream videos from UEs, we use H.264 in FFmpeg [181] 4.1.9 for video encoding and secure reliable transport (SRT) [182] for transmission. To run video analytics on the server, we use TensorFlow 2.6.2 C API + CppFlow [183] and PyTorch 1.10.1 [184] C++ API for DNN models and OpenCV 4.4.0 [185] for image processing.

Testbed setup. We evaluate Pendulum on a 5G RAN-enabled edge testbed (Figure 5.18). It consists of a commodity server equipped with Intel Xeon Gold 5128 CPU and 8 \times RTX 2080 Ti GPUs, where we run the OpenAirInterface (OAI)-5G RAN software [152] and video analytics applications. As an RF frontend, we use USRP X310 (TDD n78 band with 5DDDSU and single MIMO layer following [36,186]) connected to the server. For UEs, we use Google Pixel 6 smartphones with programmable SIM cards (sysmoISIM-SJA2) [153]. We emulate the network bandwidth changes by controlling the MCS. As the RAN stack has stability issues, including hardware compatibility and UE connection failures (similar reported in [4]), for larger-scale experiments, we use emulated UEs running in a desktop machine and use OAI-5G RF simulator [187] and

Table 5.1: Evaluation datasets.

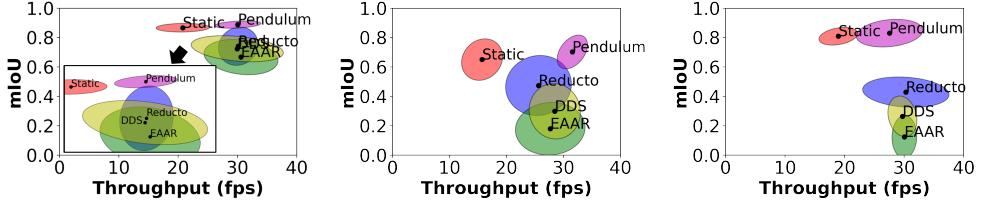
	Camera	Content	Scene Change	# Videos
MOT [34]	CCTV, Handheld	Streets	Moderate	5
BDD [157]	Dashcam	City road	Fast	6
Game (self-collected)	Dashcam	Racing game	Very fast	10

Linux `tc` [154] to emulate network bottleneck. To emulate a compute bottleneck, we linearly increase the DNN inference latency by injecting delay after inference [14].

Datasets. We use three datasets with different scene change speeds (Table 5.1): five 30fps videos (02, 04, 09, 10, 11) for MOT [34] (moderate), 6 videos from BDD [157] (fast), and self-collected racing game videos (Games) from YouTube (very fast). All videos are scaled to 720p. For videos without ground truth labels, we use golden config (heaviest backbone, highest bitrate) outputs. We observe consistent results across datasets and report the results on BDD unless specified.

Tasks and DNNs. We use two tasks: object detection and semantic segmentation. For detection, we use YOLOv5 [33] with 5 backbones (n/s/m/l/x), and EfficientDet with 7 backbones (D0-D6) [108]. For segmentation, we train FPN [168] with 7 EfficientNet backbones (B0-B6) [148] on BDD [157]. Unless stated otherwise, we report performance using EfficientDet.

Single-stage baselines: **Static** uses a fixed (bitrate, backbone). **DDS** [87] uses two-path streaming (low-quality probe frame + high-quality feedback for regions with low-confidence inference results). **EAAR** [1] uses dynamic RoI encoding (high quality only for regions where objects existed in the previous frame) and motion vector-based frame filtering. **Reducto** [78] uses pixel/edge/area feature difference-based frame filtering (feature type is offline profiled per each task). **Backbone Adaptation (BA)** only adapts the DNN backbone (based on our profiler in Chapter 5.5) and uses a fixed bitrate. This is equivalent to



(a) MOT (moderate speed) (b) BDD (fast speed) (c) Game (very fast speed)
Figure 5.17: Throughput-accuracy comparison in network bottleneck scenario.

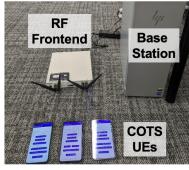


Figure 5.18:

Testbed imple-
mentation.

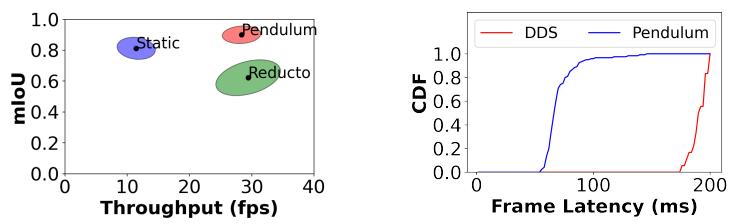


Figure 5.19: Over-the-air per-
formance.

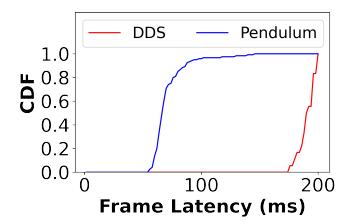


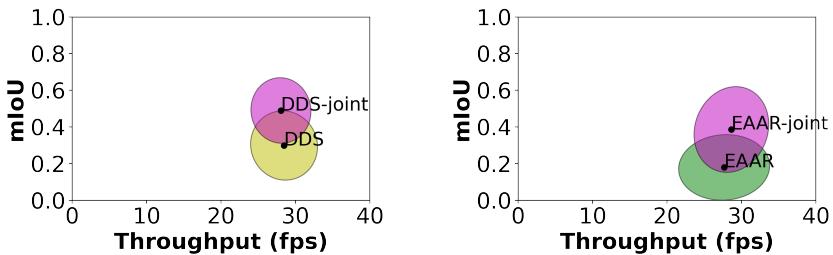
Figure 5.20: Frame-wise la-
tency comparison.

single-knob Chameleon [40].

Multi-stage baseline: Pendulum-Decoupled is a network-compute-decoupled joint scheduler. When the network becomes a bottleneck, all users reduce their bitrates by the same amount until the bottleneck is resolved. Then, the server compensates for the accuracy drop by choosing the cost-optimal backbones that satisfy their accuracy requirements.

5.7.1 End-to-End Improvement

Throughput-accuracy. Figure 5.17 compares the throughput-accuracy of Pendulum against baselines across three datasets. Ellipses show the $1-\sigma$ range of the results. We throttle the network bandwidth from 20 to 3 Mbps after 4 seconds after the streaming start. Static, which uses a fixed (4 Mbps, EfficientDet-D1), suffers from throughput drops due to network bottlenecks. Overall, Pendulum consistently achieves ≈ 30 fps throughput and higher accuracy compared to the baselines: up to 0.64 mIoU gain (Game, Pendulum: 0.81 vs. EAAR: 0.17).



(a) DDS vs. DDS-joint, BDD

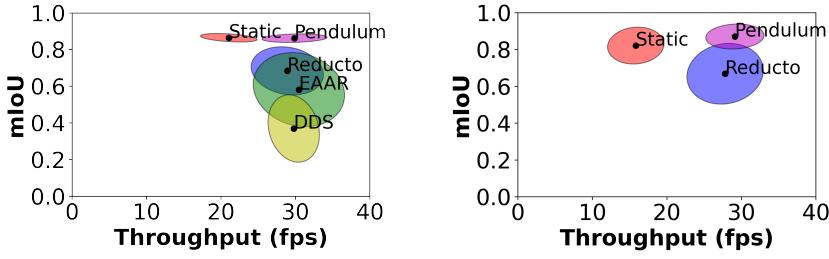
(b) EAAR vs. EAAR-joint, BDD

Figure 5.21: Joint scheduling on state-of-the-art systems.

The performance of the baselines varies depending on the dataset. For MOT (moderate scene changes), all baselines effectively optimize the bitrate to below 3 Mbps (e.g., EAAR: 2.98 Mbps), achieving comparable accuracy to Pendulum. However, for BDD and Game with faster scene changes, the accuracy drops significantly, especially for EAAR and DDS. For EAAR, the object region from the previous frame becomes highly stable. For DDS, fast-changing video encoded in a low-bitrate probe stream (e.g., 1 Mbps) suffers from severe quality degradation, resulting in inaccurate DNN inference and feedback frame requests. Consequently, both end up streaming the entire frame at low quality. Pendulum achieves higher mIoU even when DDS and EAAR use the heaviest D6 backbone (e.g., 0.71 vs. 0.49, 0.48 in BDD).

Over-the-air performance. Figure 5.19 shows the throughput-accuracy performance in the real physical channels using our USRP RAN implementation. We observe that Pendulum effectively handles network bottlenecks and achieves high throughput and accuracy compared to baselines.

Frame-wise latency. While achieving high accuracy and throughput, Pendulum also achieves low frame-wise latency. Figure 5.20 shows that Pendulum yields <100 ms frame-wise latency, much smaller than DDS which involves two frame transfers and DNN inference per frame.



(a) YOLOv5 (detection), MOT

(b) FPN (segmentation), BDD

Figure 5.22: Performance across various tasks & DNNs.

5.7.2 Joint Scheduling on SOTA Systems

We next demonstrate that joint scheduling can be generally integrated into prior state-of-the-art systems. For example, DDS [87] controls two bitrates at which the probe and feedback streams are encoded (e.g., 2 and 6 Mbps). In case of the network bottleneck, along with adjusting the two bitrates (e.g., to 1 and 3 Mbps), DDS-Joint increases the DNN backbone from EfficientDet-D1 to D6 (increment can be optimized by profiling). Figure 5.21(a) and (b) show the results on the BDD dataset. Joint scheduling achieves 0.17 and 0.20 higher mIoU than baseline DDS and EAAR, respectively, demonstrating its generalizability.

5.7.3 Performance on Various App Settings

Various DNNs. We repeat the same experiment as in Figure 5.17(a), but with YOLOv5 detector backbones. We see a similar trend: Pendulum achieves 0.17, 0.32, and 0.52 higher mIoU than Reducto, EAAR, and DDS, respectively. Note that DDS’s mIoU is lower than when using the EfficientDet backbones because the lightweight YOLOv5 backbone yields less accurate feedback regions on low-bitrate videos.

Various tasks. Figure 5.22(b) also shows that Pendulum achieves similar performance for FPN segmentation models (e.g., ≈ 30 fps throughput with 0.19 higher mIoU than Reducto).

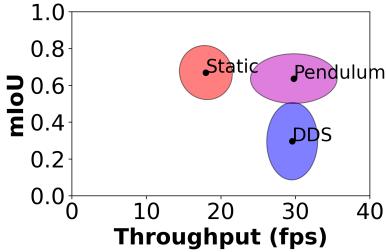


Figure 5.23: Performance for (res, backbone) knobs.

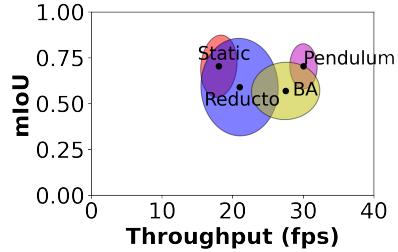


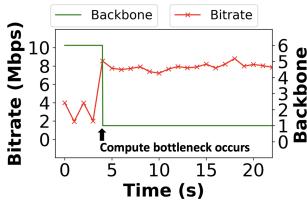
Figure 5.24: Performance in compute bottleneck.

Other scheduling knobs. Figure 5.23 shows the performance of Pendulum with (resolution, backbone) knobs: upon network bottleneck, Pendulum reduces the resolution from 720p to 540p, and increases the backbone. Pendulum achieves both high throughput and accuracy, showing the generality of joint scheduling with respect to scheduling knobs.

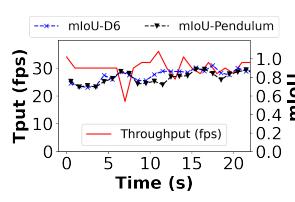
5.7.4 Performance in Compute Bottleneck

Throughput-accuracy. Figure 5.24 shows the performance of Pendulum in the compute bottleneck scenario on MOT. We increase the inference latency slowdown factor from 1 \times to 2 \times 4 seconds after the app starts. Static (2 Mbps, EfficientDet-D6) suffers from throughput drop. Compared to Reducto and BA which only reduce the DNN workload (either by reducing the number of frames or backbone), Pendulum effectively increases the bitrate (from 1.98 to 5.42 Mbps) resulting in 0.12 higher mIoU and 1.29 \times higher throughput.

Operation timeline. Figure 5.25 shows an example operation timeline of Pendulum in compute bottleneck scenario for a BDD video. Figure 5.25(a) shows the bitrate and backbone over time. At the start, Pendulum uses (2 Mbps, EfficientDet-D6) config; bitrate peaks every 2 seconds due to periodic high-bitrate frames for profiling. After 4s when compute bottleneck occurs, Pendulum reduces the backbone to D1 and increases the bitrate to 8 Mbps (according



(a) Bitrate and backbone.



(b) Throughput and mIoU.

Figure 5.25: Operation timeline when the compute becomes a bottleneck.

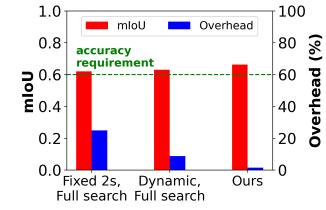


Figure 5.26: Demand profiler performance breakdown.

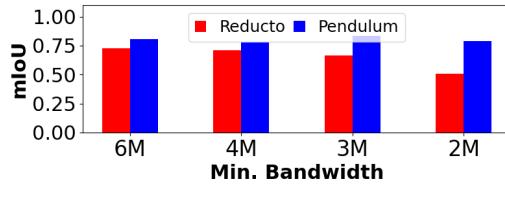


Figure 5.27: Performance under bandwidth fluctuation.

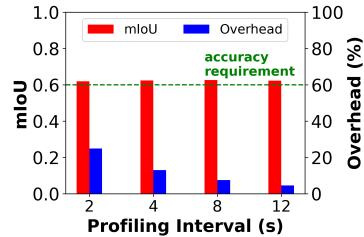


Figure 5.28: Impact of profiling interval on accuracy.

to the profiling results). Figure 5.25(b) shows the throughput and mIoU in the same timeline. Pendulum shortly suffers from throughput drop when a bottleneck occurs but quickly recovers from it while retaining the same accuracy using the D6 backbone.

5.7.5 Microbenchmarks

5.7.5.1 Robustness Under Bandwidth Fluctuation

Figure 5.27 shows the performance of Reducto and Pendulum under bandwidth fluctuation scenarios. For real-time throughput, both systems conservatively reduce the bitrate to minimum observed bandwidth to avoid sudden throughput drop. While Reducto’s accuracy quickly drops as bottleneck becomes more severe, Pendulum robustly retains the accuracy by increasing the backbone.

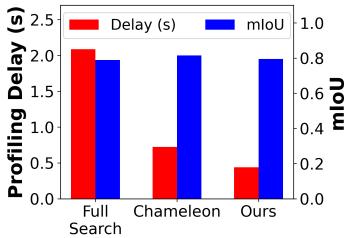


Figure 5.29: Performance of accuracy modeling.

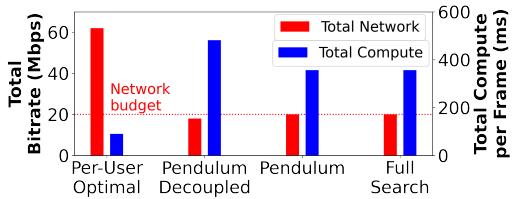


Figure 5.30: Multi-user scheduling performance.

5.7.5.2 Network-Compute Demand Profiler

Performance breakdown. Figure 5.26 shows the performance breakdown regarding profiling overhead and mIoU. Profiling overhead (%) is the ratio between the total sum of DNN inference latencies for profiling and serving. We use 5 YOLOv5 backbones (n/s/m/l/x) and 4 bitrates (1, 2, 4, 8 Mbps); a full search over all configs takes 1s. Fixed interval, full search profiling on 2 frames per every 2s window incurs 24.9% overhead. Scene change-based dynamic profiling reduces the overhead to 8.8%. Finally, leveraging the knob independence (which reduces the number of searches to 8), reduces the overhead to 1.5% (93.9% smaller than the fixed-interval, full search) with negligible mIoU drop.

Impact of profiling interval. Figure 5.28 shows the impact of profiling interval on serving accuracy and overhead. It shows the necessity of dynamic profiling: triggering the profiling intermittently at every 12s reduces overhead by 18% compared to 2s without accuracy drop. Our scene change detector accurately triggers profiling at every 9.09s on average.

Weighted multi-knob accuracy interpolation. Figure 5.29 shows the effectiveness of accuracy modeling: when profiling 4×7 configs (1,2,4,8 Mbps and EfficientDet D0-D6) on BDD, our profiler reduces the overhead by 79% and 40% compared to full search and Chameleon [40] without accuracy drop.

5.7.5.3 Iterative max-cost gradient scheduler

Figure 5.30 compares the performance of various schedulers. We assume 10 users with randomly sampled demand curves from MOT and BDD datasets (each with 4-6 Pareto-optimal configs, bitrate range in [1,10] Mbps, and EfficientDet-D0-D6 backbones). We assume a network bottleneck scenario, where the total bitrate sum should not exceed 20 Mbps. We model cost functions as linear assuming linear billing models [188,189], with unit network and compute costs are set as \$0.36 and \$0.74 (Chapter 5.3.1). *Per-User Optimal* chooses user-wise cost-optimal configs independently, resulting in severe network bottleneck. While achieving the same app accuracy requirement, our Iterative Max Cost Gradient algorithm achieves comparable performance compared to the full search optimal solution, with only 0.004% searches. It also reduces the total cost by 25% compared to **Pendulum-Decoupled**: this is because **Decoupled** reduces the bitrate of users with low-cost gradients (i.e., require large inference latency increase for accuracy compensation), whereas **Pendulum** efficiently chooses users with high-cost gradients.

Chapter 6

Heimdall: Mobile GPU Coordination Platform for AR Applications

6.1 Introduction

In this Chapter, we design **Heimdall**, a mobile GPU coordination platform to support concurrent multi-DNN and rendering tasks for AR apps. **Heimdall** newly designs and implements a *Pseudo-Preemptive mobile GPU coordinator* to enable highly flexible coordination among multi-DNN and rendering tasks. **Heimdall** is distinguished from prior work in that i) it coordinates latency-sensitive foreground rendering tasks along with background DNN tasks to achieve stable rendering performance of ≈ 30 fps, and ii) it addresses resource contention among multiple DNNs to meet their latency requirements.

Designing **Heimdall** involves the following challenges:

- **Multi-DNN GPU contention.** Compared to prior mobile deep learning frameworks [18, 20, 37, 38] that have mostly been designed for running a single DNN, emerging AR apps require concurrent multi-DNN execution (Chapter 2.1.2). Not only are the individual state-of-the-art DNNs very complex to run in real-time (Chapter 2.2.1), running multiple DNNs concurrently incurs

severe contention over limited mobile GPU resources, degrading overall performance. For example, our study shows that running 3 to 4 different DNNs commonly required in AR apps (e.g., object detection, image segmentation, hand tracking) concurrently on Google TensorFlow-Lite (TF-Lite) [37] and Xiaomi MACE [38] over high-end Adreno 640 GPU incurs as high as $19.7\times$ slowdown (Chapter 2.2.4). Although several recent studies aimed at running multiple DNNs concurrently on mobile [7, 16, 56], they have mostly focused on memory optimization [16, 56] or cloud offloading [7]; multi-DNN GPU contention remains unsolved.

- **Rendering-DNN GPU contention.** More importantly, prior works only consider a DNN running in an isolated environment where no other task is contending over the GPU. When running rendering in parallel with DNNs, GPU contention degrades and fluctuates the frame rate, degrading user experience (e.g., drops from 30 to 11.99 fps when 4 DNNs run in background (Chapter 2.2.4)).

There have been studies to schedule concurrent tasks on desktop/server GPUs [59–61, 68, 69, 71, 190, 191], either with *parallel execution* by dividing GPU cores (e.g., using NVIDIA Hyper-Q [62]) with hardware architectural support, or with *time-sharing* through preemption (e.g., using CUDA stream prioritization). However, mobile GPUs do not provide architectural support for parallel execution, while fine-grained preemption is not easy as well due to high context switch costs caused by large state size and limited memory bandwidth (Chapter 6.3.1). Even with architecture evolution, the need for an app-aware coordinator to dynamically prioritize and allocate resources between multiple DNNs persists (Chapter 7.2.3). We can also consider cloud offloading, but it is not trivial to employ it in outdoor scenarios where network latency is unstable.

To tackle the challenges, we design a *Pseudo-Preemption* mechanism to support flexible scheduling of concurrent multi-DNN and rendering tasks on

mobile GPU. We take the *time-sharing* approach as a baseline, and enable context switches only when a semantic unit of the DNN or rendering task is complete. This does not incur additional memory access cost, which is the core difficulty in applying conventional preemption (triggered by periodic hardware interrupt regardless of the app context) for mobile GPUs. Accordingly, it allows the multi-DNN and rendering tasks to time-share the GPU at a fine-grained scale with minimal scheduling overhead. With this new capability, we flexibly prioritize and run the tasks on the GPU to meet the latency requirements of the AR app. Our approach can also be useful for the emerging neural processors (e.g., NPUs or TPUs), as preempting hard-wired matrix multiplications is complicated and context switch overhead can be more costly due to larger state sizes (Chapter 7.2.3).

To implement *Pseudo-Preemption* mechanism, **Heimdall** incorporates the following components:

- **Preemption-enabling DNN analyzer.** The key in realizing *Pseudo-Preemption* is breaking down the bulky DNNs into small schedulable units. Our *Preemption-Enabling DNN Analyzer* measures the execution times of DNN and rendering tasks on the target mobile device and partitions the DNNs into the units of scheduling to enable fine-grained GPU time-sharing with minimal scheduling overhead. We notice that the execution time of individual DNN operator (op) is sufficiently small (e.g., <5 ms for 89.8% of ops). Exploiting this, the analyzer groups several consecutive ops as a scheduling unit which can fit between the two consecutive rendering events. As rendering latencies are often very small (e.g., 2.7 ms for rendering a 1080p camera frame), each task is used as the scheduling unit. Note that existing frameworks run the entire bulky DNN inference all at once (e.g., `Interpreter.Run()` in TF-Lite [192], `MaceEngine.Run()` in MACE [38]), limiting multi-DNN and rendering tasks to share the mobile GPU at a very coarse-grained scale.

- **Pseudo-preemptive GPU coordinator.** We design a GPU coordinator that schedules the DNN and rendering tasks on GPU and CPU. It can employ various scheduling policies based on multiple factors: profiled latencies, scene variations, and app/user-specified latency requirements. As the base scheduling policy, the coordinator assigns the top priority to the rendering tasks and executes them at the target frame rate (e.g., 30 fps) to guarantee the usability of the app. Between the rendering events, the coordinator decides the priority between multiple DNNs and determine which chunk of DNN ops (grouped by the analyzer) to run on the GPU. It also decides whether to offload some DNNs to the CPU in case there is a high level of contention on the GPU. Note that existing frameworks provide no means to prioritize a certain task over others, making it difficult to guarantee performance under contention.

Our major contributions are summarized as follows:

- To our knowledge, this is the first mobile GPU coordination platform for emerging AR apps that require concurrent multi-DNN and rendering execution. We believe our platform can be an important cornerstone to support many emerging AR apps.
- We design a *Pseudo-Preemption* mechanism to overcome the limitations of mobile GPUs for supporting concurrency. With the mechanism, **Heimdall** enhances the frame rate from ≈ 12 to ≈ 30 fps while reducing the worst-case DNN inference latency by up to ≈ 15 times compared to the baseline multi-threading method.
- We implement **Heimdall** on MACE [38], an OpenCL-based mobile deep learning framework, and conduct an extensive evaluation with 8 state-of-the-art DNNs (see Table2.2) and various mobile GPUs (i.e., recent Adreno series) to verify the effectiveness.

Algorithm 3 OpenCL-based DNN inference in MACE

```
1: for Operator in Graph do
2:   TargetDevice  $\leftarrow$  Operator.GetTargetDevice() TargetDevice == GPU
3:   Kernel  $\leftarrow$  Operator.GetKernel()
4:   clCommandQueue.enqueueNDRangeKernel(Kernel) TargetDevice == CPU
5:   clCommandQueue.finish()
6:   Operator.RunOnCPU()
```

6.2 Analysis on GPU Contention

The workload of upcoming AR apps is unique in that it runs multiple compute-intensive DNNs simultaneously while seamlessly rendering the virtual contents. However, existing mobile deep learning frameworks lack support for multi-DNN and rendering concurrent execution, and severe GPU contention incurs significant performance degradation for both DNN and rendering tasks.

Algorithm 3 shows the OpenCL-based DNN inference flow in MACE.¹ Upon the inference start, the framework executes a series of operators (ops) constituting the DNN. Per each op, the framework first identifies if it is executed on GPU or CPU (lines 1–2). A GPU op is executed by enqueueing its kernel to the command queue to be executed by the GPU driver (lines 2–4). As enqueueNDRangeKernel() function is an asynchronous call, consecutive GPU ops are enqueued in short intervals (few μ s) and executed in batches by the driver to enhance GPU utilization. However, when a CPU op is encountered, it can be executed only after the previously enqueued GPU ops are finished and the result is available to the CPU via CPU/GPU synchronization (lines 4–6).

Figure 6.1 illustrates an example 3-DNN GPU contention scenario that can occur in the above inference process. Following conventional mobile deep learning frameworks, each DNN inference is initiated on separate thread and infer-

¹The logic is implemented in `SerialNet.Run()` function, while TF-Lite is implemented similarly using OpenGL/OpenCL.

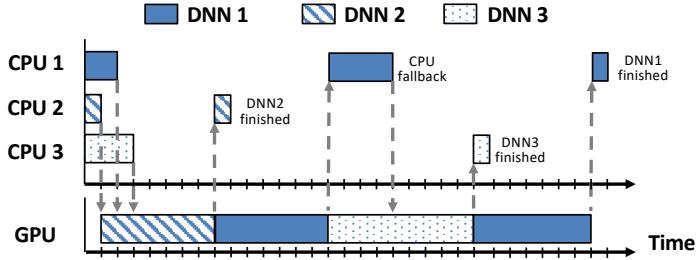


Figure 6.1: Multi-DNN GPU contention example.

ence engine. Each thread on different CPU cores first runs the input preprocessing and enqueues the DNN inference (a sequence of ops) to the GPU scheduler. The GPU scheduler processes the ops from multiple threads in FIFO. At this step the first contention occurs; DNN#1 and #3 cannot access the GPU until the already running DNN#2 is finished. After DNN#2 finishes, DNN#1 takes control over the GPU and runs its inference. However, let’s assume that some ops in DNN#1 are not supported by the GPU backend of the framework and needs to be executed on the CPU (Table 2.2 shows how frequently this occurs for different DNNs; more details are in Chapter 6.6.2). In such a case, DNN#1 encounters another contention: even when the CPU op execution is finished, it cannot access the GPU until the already running DNN#3 finishes. As a result, the inference latency of DNN#1 is significantly delayed.

The above contention becomes more severe with more number of DNNs concurrently running. Furthermore, DNNs with more CPU fallback ops suffer more from contention, as they lose access over the GPU at every CPU op execution. For example, in Figure 2.4(a), StyleTransfer [25] containing 14 CPU ops suffers the most latency overhead compared to other DNNs that contain no CPU ops.

6.3 Heimdall System Overview

6.3.1 Approach

The core challenge in supporting concurrency on mobile GPU lies in the lack of support for parallelization or preemption. As analyzed in Chapter 6.2, mobile GPU can run only a single task at a given time, making it hard to provide stable performance when multiple tasks are running. Existing mobile deep learning frameworks, however, fail to consider such limitations, and are ill-suited for AR workloads in two aspects: i) they run the entire bulky DNN inference all at once (e.g., by `Interpreter.Run()` in TF-Lite, `MaceEngine.Run()` in MACE), limiting multi-DNN and rendering tasks to share the GPU at a very coarse-grained scale (Table 2.2), and ii) they provide no means to prioritize a certain task over others, making it challenging to guarantee performance under contention.

6.3.1.1 Why Not Apply Desktop GPU Scheduling?

One possible approach is to implement parallelization or preemption in mobile GPUs. Although there have been many studies to support multitask scheduling on desktop/server-grade GPUs [59–61, 71, 190, 191], they are either designed for CUDA-enabled NVIDIA GPUs (which are unsupported in mobile devices) or require hardware modifications (e.g., memory hierarchy [67]), making it difficult to apply for commodity mobile GPUs. Also, adopting similar ideas is not straightforward due to the following limitations of mobile GPUs.

Limited architecture support. Several studies focused on spatially sharing the GPU to run multiple kernels in parallel, either by partitioning the computing resources [71, 190] (e.g., starting from Kepler architecture [193] released in 2012, NVIDIA GPUs can be parallelized in units of Streaming Multiprocessors using Hyper-Q [62]) or fusing parallelizable kernels with compiler techniques [191, 194]. However, such techniques are unsupported in mobile GPUs

architecturally at the moment.

Limited memory bandwidth. Other studies aimed at time-sharing the GPU by fine-grained context switching [59–61], as well as enabling high-priority tasks to preempt the GPU even when others are running [71] (e.g., by using CUDA stream prioritization). However, frequent context switching incurs high memory overhead due to large state size, which is burdensome for mobile GPUs with limited memory bandwidth. For example, ARM Mali-G76 GPU in Samsung Galaxy S10 (Exynos 9820) has 26.82 GB/s memory bandwidth shared with the CPU, which is 23× smaller than that of NVIDIA RTX 2080Ti (i.e., 616 GB/s). Each context switch requires 120 MB memory transfer (=20 cores×24 execution lanes/core×64 registers/lane×32 bits), which incurs at least 4.36 ms latency even when assuming the GPU fully utilizes the shared memory bandwidth. While recent Qualcomm GPUs (Adreno 630 and above) support pre-emption [195] (which can be utilized by setting different context priorities in OpenCL), we observed that each context switch (both between rendering–DNN and DNN–DNN) incurs 2–3 ms overhead on LG V50 with Adreno 640 GPU, aside from the fact that the priority scheduling is possible only at a coarse-grained scale (i.e., low, medium, and high). Such memory overhead would be burdensome in the multi-DNN and rendering AR workload, where context switch should occur at a 30 fps (or higher) scale.

6.3.1.2 Our Approach: Pseudo-Preemption

To tackle the challenges, we design a *Pseudo-Preemption* mechanism to coordinate multi-DNN and rendering tasks. As parallelization is unsupported in mobile GPUs, we take the *time-sharing* approach as a baseline. To mimic the effect of preemption while avoiding the burdensome context switch memory overhead, we divide the DNN and rendering tasks into smaller chunks (i.e., scheduling units) and switch between them only when each task chunk is fin-

ished, enabling multi-DNN and rendering tasks to time-share the GPU at a fine-grained scale. A possible downside of our approach is that fragmenting the GPU tasks may incur latency overhead, as the GPU driver would lose the chance to batch more tasks to enhance GPU utilization. However, such overhead can be minimized as we can flexibly adjust the scheduling unit size to balance time-sharing granularity and latency overhead (e.g., 89.8% of the DNN ops run within 5 ms, and rendering latencies are typically small).

6.3.2 Design Considerations

Commodity mobile device support. Our goal is to support a wide range of commodity mobile devices by requiring no modification to existing hardware or GPU drivers. We focus on using mobile GPU and CPU in this work, and plan to add NPU/TPU support when the hardware and APIs are more widely supported. We also leave cloud/edge offloading out of our scope, as it introduces latency issues in outdoor mobile scenarios.

Guarantee stable rendering performance. Our main goal is to enable seamless rendering even in the presence of multi-DNN execution. We aim to minimize the frame rate drop and fluctuation due to GPU contention, which harms the user experience.

Coordinate Multi-DNN execution. While guaranteeing seamless rendering, we aim to coordinate multiple DNNs to satisfy the app requirements with minimal inference latency overhead.

No loss of model accuracy. Our goal is to incur no accuracy loss for each DNN inference. We leave runtime model adaptation for latency-accuracy trade-off (e.g., via pruning [56]) to future work.

Transparency. Finally, we aim to design a system that minimizes the extra efforts required for the app developers to use our platform.

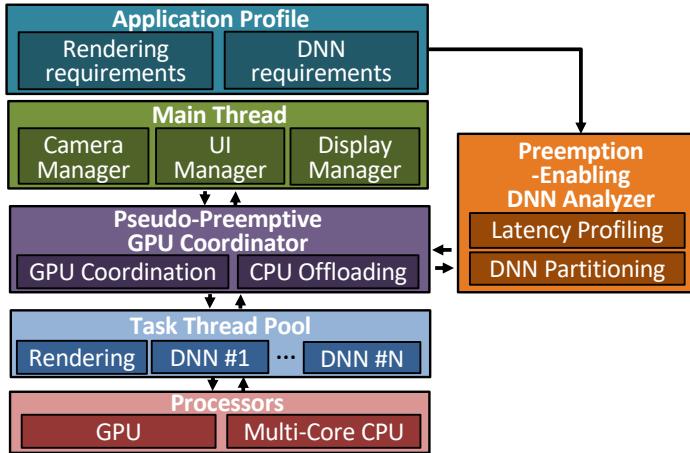


Figure 6.2: System Architecture of Heimdall.

6.3.3 System Architecture

Figure 6.2 depicts the overall architecture of Heimdall. Heimdall is located on the framework layer to determine the scheduling units of the DNNs provided by the application, and coordinate their execution based on app-specified priorities and deadlines. It does not require any fix on the OS GPU scheduler. Given the app profile (rendering frame rate and resolution, DNNs to run and latency constraints), *Preemption-Enabling DNN Analyzer* first profiles the information necessary to determine the scheduling units to enable the *Pseudo-Preemption* mechanism. First, it profiles the rendering and DNN inference latencies on the target AR device to determine how much time the DNNs can occupy the GPU between the rendering events (Chapter 6.4.2). Second, it partitions the DNNs into chunks (scheduling unit) that can fit between the rendering events with minimal inference latency overhead (Chapter 6.4.3).

At runtime, *Pseudo-Preemptive GPU Coordinator* takes multi-DNN and rendering tasks from the main thread (that controls the camera, UI, and display), and coordinates their execution to satisfy the app requirements. Specifically, it first defines a *utility function* to compare which DNN is more important to run at a given time based on the inference latency and scene contents

(Chapter 6.5.2), and coordinates their execution on GPU, as well as dynamically offload some DNNs to the CPU to reduce the GPU contention (Chapter 6.5.3).

6.4 Preemption-Enabling DNN Analyzer

6.4.1 Overview

What should we analyze? The goals of the analyzer are i) profile rendering and DNN inference latencies on the target device (which varies depending on the mobile SoC and GPU) to let the coordinator get a grasp on how it can dynamically schedule their execution, and ii) partition the bulky DNNs into chunks (i.e., the units of scheduling), to enable fine-grained GPU coordination and guarantee rendering performance.

Static profiling vs. dynamic profiling? The app requires to run multi-DNN, rendering, and other tasks (e.g., pre/postprocessing for the DNN inference, camera) simultaneously, which may fluctuate the execution times of each task at runtime. However, as mobile GPUs do not support preemption (i.e., a task cannot be interrupted once started), the execution times on GPU remain stable regardless of the presence of other tasks. Thus, offline profiling and DNN partitioning approach is feasible for GPU. However, the execution times of DNNs on CPUs may fluctuate due to resource contention; Figure 6.3 shows that the inference times on CPU increase and fluctuate when the camera is running in background. Thus, CPU execution times need to be continuously tracked at runtime.

How fine should we partition the DNNs? Inference times of DNNs typically exceed multiple rendering intervals as shown in Table 2.2. At the op-level, however, the execution times remain small enough, making fine-grained partitioning feasible to fit in between the rendering events. For example, for the 7 DNNs in Table 2.2 whose inference latencies are over 33 ms, Figure 6.4 shows

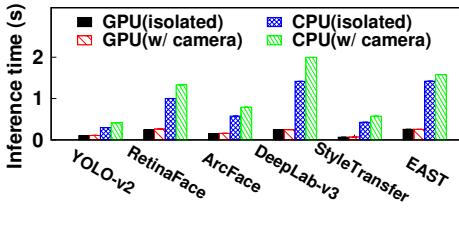


Figure 6.3: DNN inference latency with and without camera.

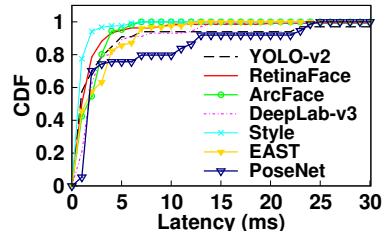


Figure 6.4: Operator-level latency distribution.

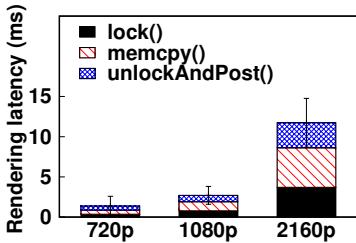


Figure 6.5: Camera frame rendering latency.

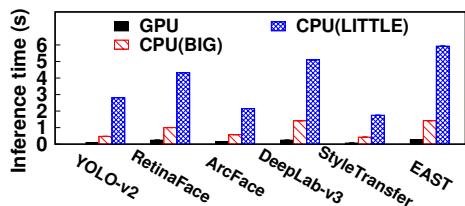


Figure 6.6: Example DNN latency profiling result on Google Pixel 3 XL.

that on average 89.8% of the ops run within 5 ms on Google Pixel 3 XL. Therefore, it suffices to partition the DNNs at the op-level and not below (e.g., convolution filter-level). However, note that dividing the DNN too finely also has its downside: it incurs higher latency overhead as the GPU driver loses the chance to batch more consecutive ops to enhance GPU utilization.

6.4.2 Latency Profiling

Rendering latency. Given the target rendering frame rate (f) and input video resolution, the analyzer first measures the rendering latency, T_{render} . This determines how much time the DNNs can occupy the GPU between rendering events (i.e., $\frac{1}{f} - T_{render}$). For example, rendering 1080p frames on Adreno 640 GPU in LG V50 takes 2.7 ms (Figure 6.5), leaving 30.6 ms for DNNs when the frame rate is 30 fps.

DNN latency. Secondly, the analyzer measures the DNN inference latencies

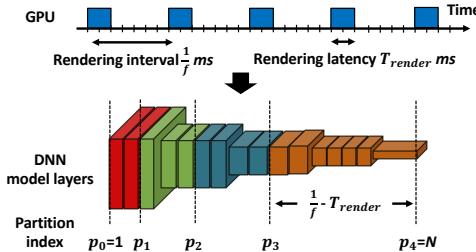


Figure 6.7: Operation of DNN partitioning.

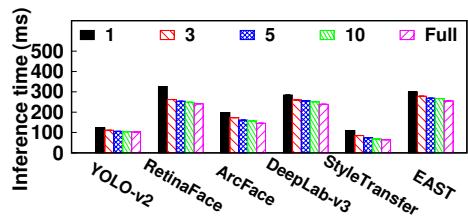


Figure 6.8: DNN inference latencies for varying partition sizes.

on the target GPU and CPU. Figure 6.6 shows an example of the profiled results on different processors (i.e., the GPU and CPU cores in the ARM big.LITTLE architecture) in LG V50.² The analyzer also measures the inference latencies of DNNs running on CPU at runtime to track variations due to CPU resource contention.

6.4.3 DNN Partitioning

Basic operation. Figure 6.7 shows the operation of DNN partitioning. Given a DNN D composed of N ops, let $T(D_{i,j})$ denote the execution time of a subgraph from i -th to j -th op. Our goal is to determine a set of K indices $\{p_1 = 1, p_2, p_3, \dots, p_K = N\}$ that partition the DNN in a way such that each partition execution time lies within the rendering interval,

$$T(D_{p_i, p_{i+1}}) \leq \frac{1}{f} - T_{render} \quad 1 \leq i \leq K - 1. \quad (6.1)$$

Although there are multiple solutions that satisfy the constraints, dividing the model too finely (e.g., running only one or two ops at a time) incurs higher scheduling overhead, as the GPU driver loses the chance to batch more consecutive ops to enhance GPU utilization: Figure 6.8 shows that executing only a single op at a time incurs 13 to 70% latency overhead compared to running the

²We currently assume that each DNN uses only a single CPU core, and leave multi-core CPU execution to future work.

entire model at once. Thus, the analyzer minimizes K by grouping as many consecutive ops as possible without exceeding the rendering interval. This is done as follows: i) starting from the first op of the model, incrementally increase the op index i until the latency of executing op 1 to i exceeds the rendering interval, ii) group op 1 to $i - 1$ as the first partition, and iii) start from op i and repeat the process until reaching the final op.

Relaxation. The main drawback of our approach is that undesirable GPU idle time occurs when a partition execution time is shorter than the rendering interval (especially at the end of the model where there are not enough ops left). To alleviate the issue, we relax the constraint in Equation (6.1) and allow the partition execution time to exceed the rendering interval by a small margin (e.g., 5 ms), so that more ops can be packed to maximize GPU utilization.

6.5 Pseudo-Preemptive GPU Coordinator

6.5.1 Overview

Where does the coordinator operate? The coordinator should take into account the rendering and DNN requirements of the app, and coordinate their execution (in the units of scheduling determined by the analyzer) considering the task priorities. With this requirement, we embed the coordinator in app-level deep learning framework, rather than the OS or the device driver layer where the workloads are highly abstracted.

Operational flow. The coordinator assigns the top priority to the rendering task and executes it at the target frame rate. We take this design decision as degradation or fluctuation in the rendering frame rate immediately affects the usability of AR apps. It is possible to change the scheduling policy to make rendering and DNN tasks to have the same priority in case rendering is less important.

The coordinator takes in the DNN inference requests from the main thread via admission control, so that the inference of a DNN is enqueued only after its previous inference has finished. When a DNN inference is enqueued, the latest camera frame is fed as input after either resizing it to the model input size or cropping the sub-region depending on the task. The scheduling event is triggered after every rendering event to decide the priority between DNNs and determine which DNN chunk (partitioned by the analyzer) to run on the GPU until the next rendering event. To achieve the goal, we define a *utility function* that characterizes the priority of a DNN and formulate a scheduling problem that enables fine-grained GPU time-sharing between multiple DNNs to satisfy the app requirements. It also decides whether to offload some DNNs to the CPU in case the GPU contention level is too high.

6.5.2 Utility Function

To schedule multiple DNNs, we need a formal way to compare which DNN is more important to run at a given time. For this purpose, we define a *utility function* for each DNN. The utility of a DNN D_i whose k -th inference is enqueued by the main thread at $t_{start,k}^i$ is modeled as a weighted sum of the two terms,

$$U_{D_i}(t) = L_{D_i}(t, t_{start,k}^i) + \alpha \cdot C_{D_i}(t_{start,k}^i, t_{start,k-1}^i), \quad (6.2)$$

where $L(t, t_{start})$ is the *latency utility* that measures the freshness of the inference, $C_{D_i}(t_{start,k}^i, t_{start,k-1}^i)$ is the *content variation utility* that captures how rapidly the scene content has changed from the last DNN inference, and α is the scaling factor (empirically set as 0.01 in our current implementation).

6.5.2.1 Latency Utility

The latency utility of the DNN D_i is calculated as,

$$L_{D_i}(t, t_{start,k}^i) = L_{D_i}^0 - \left(\beta_i \cdot (t - t_{start,k}^i)^{\gamma_i} \right)^2. \quad (6.3)$$

The latency utility is modeled as a concave function so that it decreases more rapidly over time to prevent the coordinator from delaying the execution too long. Three parameters can be configured to set the priorities between DNNs. β_i controls the proportion of the GPU time each DNN can occupy (e.g., setting β_i to 1 for all DNNs will enable equal sharing). $L_{D_i}^0$ and γ_i controls the priority among DNNs; a DNN with higher $L_{D_i}^0$ and γ_i will have higher initial utility but decrease more rapidly, so that the coordinator can allow it to preempt the GPU more frequently before its utility drops.

6.5.2.2 Content Variation Utility

The content variation utility D_i is computed as the difference between the input frames of the consecutive inferences at $t_{start,k}^i$ and $t_{start,k-1}^i$. Normally, this can be done by calculating the structural similarity (SSIM) [196] between the two frames. However, this is infeasible in mobile devices due to high computational complexity. Alternatively, we take the approach in [197] and compute the difference between the Y values (luminance) Y^k of the two frames (which has a high correlation with the SSIM and requires only $O(N)$ computations),

$$C_{D_i}(t_{start,k}^i, t_{start,k-1}^i) = \sum_{h=1}^H \sum_{w=1}^W |Y_{h,w}^k - Y_{h,w}^{k-1}|, \quad (6.4)$$

where H, W is the height and width of the frame.

6.5.3 Scheduling Problem and Policy

Given the DNNs and their utilities, the coordinator schedules their execution to maximize the overall performance (defined as a policy). Specifically, the coordinator operates in a two-step manner: i) schedule DNNs to efficiently share the GPU, and ii) determine whether to offload some DNNs to the CPU to resolve contention.

6.5.3.1 GPU Coordination Policy

Among many possible policies, we define two common GPU coordination policies, following a similar approach in [56]. Assume that N DNNs D_1, \dots, D_N are running on GPU, with latency constraints $t_{1,max}, \dots, t_{M,max}$ (which are set appropriately depending on the app scenario). The two policies are formulated as follows.

MaxMinUtility policy tries to maximize the utility of a DNN that is currently experiencing the lowest utility. This is done by solving,

$$\begin{aligned} & \min_i U_{D_i}(t) . \\ \text{s.t. } & t_{end,k}^i - t_{start,k}^i \leq t_{i,max} \end{aligned} \tag{6.5}$$

Under the MaxMinUtility policy, the coordinator tries to fairly allocate GPU resources to balance performance across multiple DNNs. We expect this policy to be useful in AR apps mostly consisted of continuously executed DNNs that need to share the GPU fairly (e.g., augmented interactive workspace scenario in Table 2.1).

MaxTotalUtility policy tries to maximize the overall sum of utilities of the DNNs. This is done by solving,

$$\begin{aligned} & \max_i \sum_{t=1}^N U_{D_i}(t) . \\ \text{s.t. } & t_{end,k}^i - t_{start,k}^i \leq t_{i,max} \end{aligned} \tag{6.6}$$

Under the MaxTotalUtility policy, the coordinator favors a DNN with higher utility (i.e., allow it to preempt the GPU more frequently) and runs the remaining DNNs at the minimum without violating their deadline. This policy will be useful in case an AR app requires to run high-priority event-driven DNNs at low response time (e.g., immersive online shopping scenario in Table 2.1).

6.5.3.2 Opportunistic CPU Offloading

As the app runs more DNNs in parallel, the computational complexity may exceed the mobile GPU capabilities. In such a case, GPU contention would degrade the overall utilities of the DNNs, possibly making it impossible to satisfy the app requirements. The coordinator periodically determines if some DNNs should be offloaded to the CPU to reduce the GPU contention level.

Let P_1, P_2, \dots, P_N denote the processor (GPU or CPU) the N DNNs are running on. The processor mapping is determined by solving the following problem,

$$\max_{P_1, P_2, \dots, P_N} \sum_{t=1}^N U_{D_i, P_i}(t), \quad (6.7)$$

where $U_{D_i, P_i}(t)$ denotes the utility of D_i running on processor P_i (affected by the inference time on P_i , which is profiled by the analyzer). As changing the target processor (i.e., allocating memory for the model weights and feature maps) incurs around 50 ms latency in MACE, we reconfigure the mapping at every 1-second interval.

6.5.4 Greedy Scheduling Algorithm

Solving the above scheduling problem is computationally difficult, as well as infeasible to plan offline (as the solution varies depending on scene contents). Thus, we solve it in a greedy manner to obtain an approximate solution.

GPU Coordination. For each scheduling event, the coordinator first checks how many partitions are left to execute for each DNN. Based on the profiled latencies of the remaining partitions, the coordinator checks if the inference can finish within the time left before its deadline; in case a DNN is not expected to finish within the deadline, the coordinator runs it immediately. If otherwise, the coordinator determines which DNN to execute based on their current utility values. Specifically, the MaxMinUtility policy selects a DNN with the current

lowest utility. The MaxTotalUtility policy iteratively computes the expected sum of utilities at the current scheduling event assuming that a specific DNN chunk is executed, and selects the chunk which maximizes the sum (without consideration of the future). Specifically, the utility sum is estimated by adding the latency delay equal to the scheduling interval to the latency utility of the DNNs that are not chosen, so as to reflect the additional latency delay due to the execution of another DNN.

CPU Offloading. Among the DNNs running on GPU, the coordinator picks the DNN experiencing the highest latency and offloads it to CPU if the profiled CPU inference time is $(1+m) \times$ smaller than the current latency on GPU (m is a positive margin to avoid ping-pong effect between CPU and GPU); per each scheduling event, only one DNN is offloaded to the CPU. If no DNN is offloaded, the coordinator also checks whether it should bring a DNN on CPU back to GPU. Similarly, a DNN is reloaded to GPU if its inference time on CPU is $(1+m) \times$ larger than its last inference time on GPU.

6.6 Additional Optimizations

The end-to-end inference pipeline for every DNN involves several steps that need to be executed on the CPU: i) preprocessing the input image before the inference, ii) postprocessing the inference output to an adequate form, and iii) ops in the model that are unsupported by the GPU backend of the mobile deep learning framework and needed to be executed on CPU. Granting GPU access to a DNN that currently needs to run such steps incurs unwanted GPU idle time, slowing down the overall inference latency. This becomes especially significant when processing high-resolution complex scene images. For example, RetinaFace [2] detector with inference pipeline shown in Figure 6.9 spends 106 out of 287 ms total inference time on CPU to process a 1080p image with 20

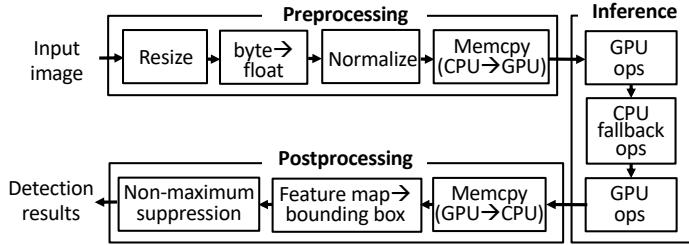


Figure 6.9: End-to-end DNN inference pipeline example for RetinaFace [2] detector.

faces. To enhance GPU utilization, we parallelize the following components.

6.6.1 Preprocessing and postprocessing

Before enqueueing a DNN inference to the task queue for the Pseudo-Preemptive GPU Coordinator to schedule, we run the following steps in parallel with other DNN inference running on the GPU), so that the DNN can fully occupy the GPU when given the access from the coordinator.

Preprocessing. The preprocessing steps involve resizing the input frame (RGB byte array) to the DNN’s input size, converting it to float array, and scaling the pixel values (e.g., from [0,255] to [-1,1]).

Postprocessing. The postprocessing steps involve converting the inference output to task-specific forms. For example, face detection requires converting the output feature map to bounding boxes and performing non-maximum suppression to filter out redundant ones.

6.6.2 CPU Fallback Operators

GPU backend of a mobile deep learning framework typically supports only a limited number of ops (i.e., a subset of the ops supported in the cloud framework). In case an op is unsupported by the GPU backend, it falls back to CPU for execution. We identify the CPU fallback op indexes of a DNN at the profil-

ing stage and run them in parallel with other DNNs at runtime. Note that CPU fallback occurs frequently, especially for complex state-of-the-art DNNs. For example, TF-Lite does not support `tf.image.resize()` required in feature pyramid network [31], which most state-of-the-art object detectors rely on for detecting small objects. Similarly, MACE does not support common ops such as `tf.crop()`, `tf.stack()`.

6.7 Implementation

We implement `Heimdall` by extending MACE [38], an OpenCL-based mobile deep learning framework, to partially run a subset of the ops in the DNN at a time by modifying the `MaceEngine.Run()` functions (and underlying functions) to `MaceEngine.RunPartial(startIdx, endIdx)`. We use OpenCV Android SDK 3.4.3 for camera and image processing. We evaluate `Heimdall` on two commodity smartphones: LG V50 (Qualcomm Snapdragon 855 SoC, Adreno 640 GPU) running on Android 10.0.0 and 9.0.0, and Google Pixel 3 XL (Snapdragon 845 SoC, Adreno 630 GPU) running on Android 9.0.0. We also used two different vendor-provided OpenCL libraries obtained from LG V50 and Google Pixel 2 ROMs. We achieved consistent results across different settings, and report the best results on LG V50.

We choose the DNNs with sufficient model accuracy for the evaluation, implement and port them on MACE (the list is summarized in Table 2.2). We implement RetinaFace [2], ArcFace [21], EAST [17], PoseNet [24] using TensorFlow 1.12.0. For MobileNet-v1 [28], CPM [27], and StyleTransfer [25], we use the models provided in the MACE model zoo [198]. For DeepLab-v3 [22] and YOLO-v2 [23], we use the pre-trained models from the original authors.

6.8 Evaluation

6.8.1 Experiment Setup

Scenarios. We evaluate **Heimdall** for 3 scenarios in Table 2.1 with the DNNs in Table 2.2: immersive online shopping, augmented interactive workspace, and AR emoji.

Evaluation metrics. We evaluate **Heimdall** with following metrics.

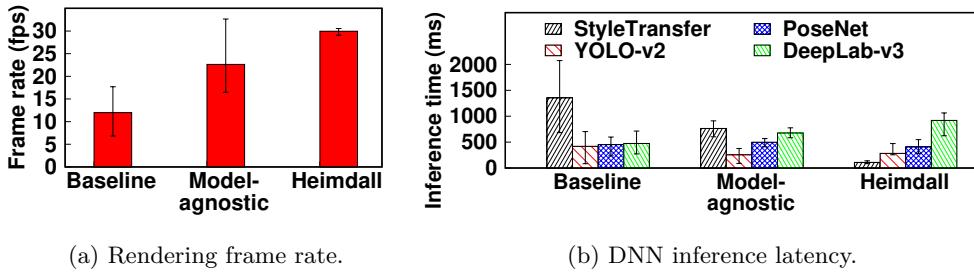
- **Rendering frame rate:** the number of frames rendered on the screen, measured every 1/3 seconds.
- **Inference latency:** the time interval between when the DNN inference is enqueued to the coordinator (after preprocessing), and when the last op of the model is executed. While we omitted pre/postprocessing latency to evaluate only the GPU contention coordination performance, end-to-end latency can also be enhanced as we parallelize such steps as well (Chapter 6.6).

Comparison schemes. We compare **Heimdall** with the following baselines.

- **Baseline MACE** creates multiple MaceEngine instances (one per each DNN) in separate threads and runs multi-DNN and rendering tasks in parallel without any coordination.
- **Model-agnostic DNN partitioning** executes 5 ops of a DNN at a time (regardless of the model or rendering requirements). This is supported in MACE to enhance UI responsiveness by preventing DNNs from occupying the GPU for too long, implemented by invoking `cl::Event.wait()` after 5 `clEnqueueNDRangeKernel()` calls.

6.8.2 Performance Overview

We first evaluate **Heimdall** with the MaxTotalUtility policy on immersive online shopping scenario compared with alternatives. The app requirements are set to render frames at 30 fps, run segmentation (DeepLab-v3) and hand track-



(a) Rendering frame rate.

(b) DNN inference latency.

Figure 6.10: Performance overview of Heimdall on LG V50.

ing (PoseNet) at 1 and 2 fps, respectively. Image style transfer (StyleTransfer) is set to have higher priority than others to satisfy the low response time requirement.

Figure 6.10(a) shows the rendering performance, where the error bar denotes the minimum and maximum frame rates. **Heimdall** supports a stable 29.96 fps rendering performance, whereas the baseline suffers from low and fluctuating frame rate (6.82-17.70 fps, 11.99 on average). While the model-agnostic partitioning slightly enhances the frame rate, it still suffers from fluctuation due to the uncoordinated execution of DNNs and rendering.

Figure 6.10(b) shows the DNN latency results, where the error bar denotes the minimum and maximum inference latencies. Overall, **Heimdall** efficiently coordinates the DNNs to satisfy the app requirements: StyleTransfer, PoseNet, and DeepLab-v3 run at 109, 409, 919 ms on average, respectively (maximum 139, 548, 1064 ms), while the worst-case inference latency of StyleTransfer is also reduced by 14.92× (from 2074 to 139 ms). This is achieved by i) giving preemptive access to StyleTransfer, ii) running DeepLab-v3 at the minimum and PoseNet more frequently to satisfy the latency constraints of both tasks, and iii) offloading YOLO-v2 to CPU to reduce GPU contention level (which also benefits YOLO-v2). Baseline and model-agnostic partitioning that cannot support such coordination fail to satisfy the app requirements, especially for StyleTransfer which is more vulnerable to GPU contention due to several CPU fallback ops as analyzed in Chapter 6.2.

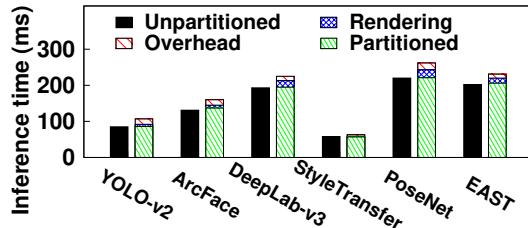


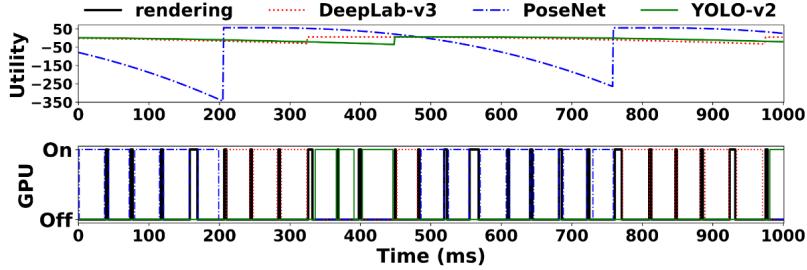
Figure 6.11: DNN partitioning overhead.

6.8.3 DNN Partitioning/Coordination Overhead

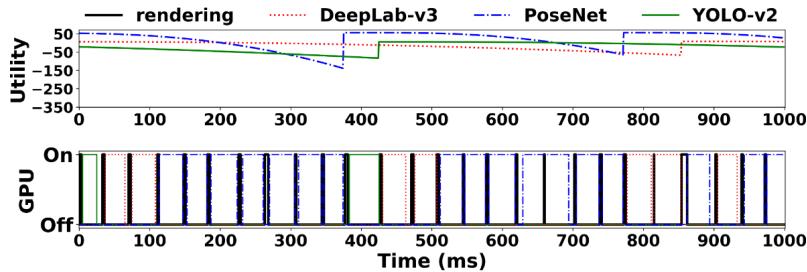
Next, we evaluate the DNN partitioning and coordination overhead on inference latency when executed with 1080p camera frame rendering at 30 fps. Figure 6.11 shows that the total GPU latency of the partitioned DNN chunks remain almost identical to unpartitioned inference latency, as Preemption-Enabling DNN Analyzer tries to pack as many ops as possible. The remaining overhead other than the rendering latency includes multiple factors, including the GPU idle time due to DNN chunks that do not perfectly fit into the rendering interval, scheduling algorithm solver, and logging process for the evaluation (this is negligible on runtime). Most importantly, our current implementation is limited to coordinating multiple DNN inferences on CPU (due to fallback or offloading) on different cores; other tasks (e.g., camera, pre/postprocessing steps) may interfere and cause latency overhead. We plan to handle the issue in our future work for further optimization.

6.8.4 Pseudo-Preemptive GPU Coordinator

GPU coordination policy. Figure 6.12 shows how the 3 DNNs in the immersive online shopping scenario are coordinated (i.e., utility over time and GPU occupancy) on the GPU under two policies in Chapter 6.5.3. Figure 6.12(a) shows that the MaxMinUtility policy executes a DNN with the currently lowest utility and enables a fair resource allocation between the 3 DNNs. Figure 6.12(b) shows that MaxTotalUtility policy favors PoseNet which has higher



(a) MaxMinUtility.



(b) MaxTotalUtility.

Figure 6.12: Performance comparison of GPU coordination policies.

priority than others (i.e., higher $L_{D_i}^0$ and γ_i in Equation (6.3), meaning that the utility is higher when the inference is enqueued but decays rapidly over time) to maximize the total utility. As a result, the utility of PoseNet remains higher than that under the MaxMinUtility policy.

Opportunistic CPU offloading. Next, we incorporate the opportunistic CPU offloading in the same setting as in Figure 6.12(a). Figure 6.13 shows the GPU/CPU occupancy and utility over time for the 3 DNNs. When CPU offloading is triggered at around $t=1600$ ms, YOLO-v2 (which had the least priority and thus had been executed sporadically) is offloaded to CPU. This benefits the other two DNNs on GPU as the contention level decreases (notice that the utility of PoseNet becomes higher after CPU offloading), while YOLO-v2 also benefits as it experiences faster inference latency as compared to when it was contending with the other two DNNs on GPU.

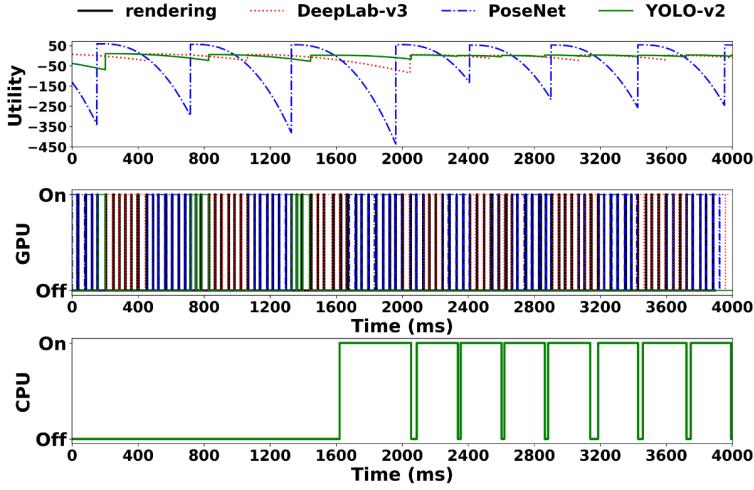


Figure 6.13: Opportunistic CPU offloading performance.

6.8.5 Performance for Various App Scenarios

Figure 6.14 shows the performance of **Heimdall** on two different scenarios: augmented interactive workspace and AR emoji. Overall, we observe consistent results. Figure 6.14(a) shows that **Heimdall** enables higher and stable rendering frame rate. Figure 6.14(b) shows that for the interactive workspace scenario, **Heimdall** coordinates the two DNNs by offloading the text detection (EAST) to the CPU so that the hand tracking (PoseNet) can run more frequently on the GPU. However, the latency gain is not as high as expected due to the scheduling overhead caused by multiple concurrent CPU tasks. Finally, Figure 6.14(c) shows that for the AR emoji scenario, **Heimdall** prioritizes StyleTransfer to guarantee low inference latency, while balancing the latencies between RetinaFace and DeepLab-v3.

6.8.6 DNN Accuracy

We evaluate the impact of **Heimdall** on DNN accuracy for the AR emoji scenario. For repeatable evaluation, we sample 5 videos of a single talking person from the 300-VW dataset [199]. As the dataset does not provide the face bounding

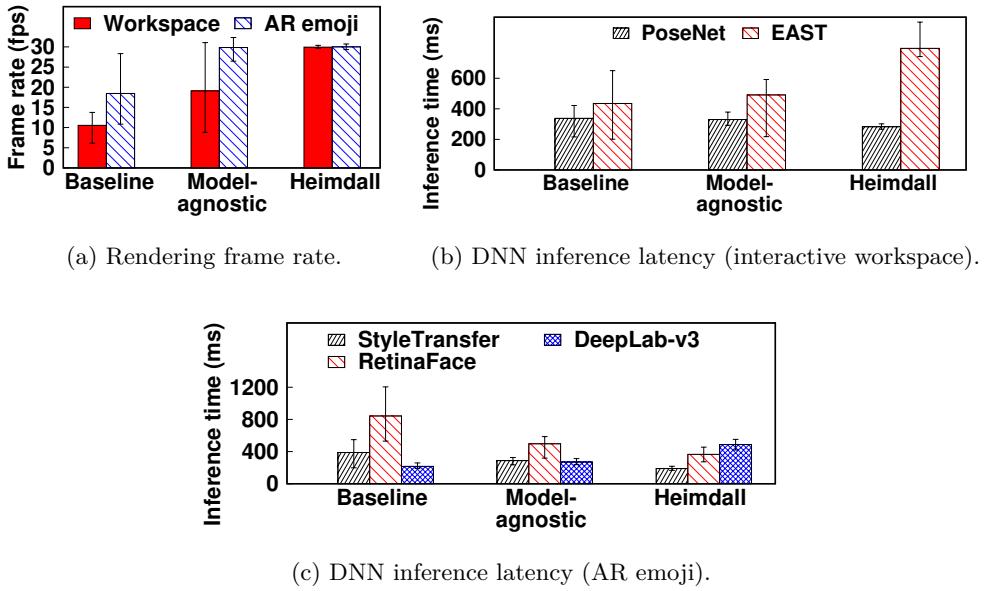


Figure 6.14: Performance of **Heimdall** for other AR app scenarios.

Table 6.1: Face detection and person segmentation accuracy (IoU) for the AR emoji scenario.

Baseline		Model-agnostic		Heimdall	
Bounding box	Mask	Bounding box	Mask	Bounding box	Mask
0.52±0.12	0.93±0.02	0.57±0.12	0.92±0.02	0.63±0.11	0.90±0.03

box and person segmentation mask labels, we run our DNNs on every frame and use the results as ground truth to be compared with the runtime detection results. Table 6.1 shows the detection accuracy in terms of mean Intersection over Union (IoU). For baseline multi-threading, face detection accuracy remains low, as RetinaFace (with several CPU fallback ops) runs at only ≈ 1 fps due to contention with DeepLab-v3 (Figure 6.14(c)). While model-agnostic partitioning alleviates the issue, it cannot coordinate the two DNNs. With **Heimdall**, we can flexibly run RetinaFace more frequently (≈ 3 fps) to improve the face detection accuracy at the cost of relatively smaller loss in the segmentation

accuracy. Note that the performance gain came from utilizing the app-specific content characteristics (i.e., the face moves more rapidly than the body). For other app scenarios, we can similarly take into account the target scene content characteristics to coordinate multiple DNNs and improve the overall accuracy.

6.8.7 Energy Consumption Overhead

Finally, we report the impact of **Heimdall** on energy consumption. We use Qualcomm Snapdragon Profiler [200] to measure the system-level energy consumption. For all the three evaluated app scenarios, baseline multi-threading consumes 4.8–5.1 W, mostly coming from the \approx 100% GPU utilization which is known to be the dominant source of mobile SoC energy consumption [201] (capturing 1080p camera frames and rendering them on screen without any DNN running consumes 1.9–2.3 W). Similarly, the GPU utilization in **Heimdall** remains \approx 100% and consumes 5.1–5.2 W. The slight increase in the energy consumption comes from the additional CPU tasks coming from the increased frame rate and the scheduling overhead of the Pseudo-Preemptive GPU coordinator.

Chapter 7

Conclusion

7.1 Summary

In this dissertation, we depicted emerging live video analytics app scenarios and characterized their workload. We then analyzed the technical challenges in realizing them, and introduced our research vision and systems to develop an end-to-end edge-cloud cooperative platform to support the workload.

We first designed **EagleEye**, a wearable camera-based system to identify missing person(s) in large, crowded urban spaces in real-time. To further innovate the performance of the state-of-the-art face identification techniques on LR face recognition, we designed a novel ICN and a training methodology that utilize the probes of the target to recover missing facial details in the LR faces for accurate recognition. We also develop Content-Adaptive Parallel Execution to run the complex multi-DNN face identification pipeline at low latency using heterogeneous processors on mobile and cloud. Our results show that ICN significantly enhances LR face recognition accuracy (true positive by 78% with only 14% false positive), and **EagleEye** accelerates the latency by 9.07 \times with only 108 KBytes of data offloaded to the cloud.

We next designed **Pendulum**, an end-to-end live video analytics system with

network-compute joint scheduling. To overcome the limitations of single-stage scheduling systems in alternating resource bottleneck scenarios, we leverage the interplay between the video bitrate and DNN complexity. Based on this, we design **Pendulum** composed of (i) a joint scheduling mechanism (to estimate network, compute resource demands and availabilities as well as control resource usages), and (ii) joint resource scheduler. Our evaluation results show that **Pendulum** achieves up to 0.64 mIoU gain (from 0.17 to 0.81) and 1.29 \times higher throughput compared to baselines.

Finally, we designed **Heimdall**, a mobile GPU coordination platform for emerging AR apps. To coordinate multi-DNN and rendering tasks, the Preemption-Enabling DNN Analyzer partitions the DNN into smaller units to enable fine-grained GPU time-sharing with minimal DNN inference latency overhead. Furthermore, the Pseudo-Preemptive GPU Coordinator flexibly prioritizes and schedules the multi-DNN and rendering tasks on GPU and CPU to satisfy the app requirements. **Heimdall** efficiently supports multiple AR app scenarios, enhancing the frame rate from 11.99 to 29.96 fps while reducing the worst-case DNN inference latency by up to \approx 15 times compared to the baseline multi-threading approach.

7.2 Discussion

7.2.1 Scalability and Generality of EagleEye to Other Workloads

Scalability to multi-person identification scenarios. While we designed **EagleEye** for single person identification scenarios, we expect that it can be easily scaled to identifying multiple targets. The only change required is extending the Identity-Specific Fine-Tuning of ICN towards multiple targets. We expect that the fine-tuning process in Chapter 4.5.2 can be applied to the aggregated probe datasets of multiple targets (e.g., 30 reference photos per each target as

evaluated in Chapter 4.8.3), as ICN has enough capacity to overfit to multiple targets (c.f., it is originally trained on FFHQ [143] with 70K identities). However, elaborating the training process would also help; for example, additional feature difference loss on the generated output face images to distinguish similar-looking targets.

Generality to other analysis workloads. The workload of other futuristic multi-DNN-enabled live video analytics applications is similar to **EagleEye**'s face identification pipeline in that they require running a series of complex DNNs repetitively to detect objects in a high-resolution scene image and analyze each identified instance (e.g., surveillance CCTVs running action recognition on individual person for crime/anomaly monitoring, Mixed Reality telepresence application where the AR glasses estimates each person's pose in its field of view and generating virtual avatar motion in the recipient's VR headset). For such workloads, **EagleEye**'s Content-Adaptive Parallel Execution can be generally adapted to enhance performance by applying different pipeline depending on the content and parallelizing the execution over heterogeneous processors on mobile and cloud.

7.2.2 Generality of Pendulum to Wider Network and System Environments

Joint Scheduling Knob Choices Under Other System Goals. We choose the knobs mainly to maximize control independency. However, other knob choices can also be feasible under different goals. For example, to minimize GPU memory overhead for edge device deployment, quantization can be considered with some model preparation efforts. When handling a wide range of network bandwidths including extreme bottleneck (e.g., < 1 Mbps for LPWAN-based disaster monitoring [202]), enhancement can be an adequate knob.

Extension to edge-cloud collaborative inference systems. Joint schedul-

ing can be extended to a collaborative inference context, i.e., across (i) on-device, (ii) network, and (iii) cloud stages. For example, in case of a compute bottleneck, the mobile device can partially run the DNN inference workload and offload the remaining to the cloud (e.g., partial RoIs [7, 99] or DNN intermediate features [14, 100, 101]), which can be jointly scheduled at the network and the cloud stages.

Extension to RAN RB scheduling. We can further improve joint scheduling efficiency if the video analytics operator also has control over RAN RB scheduler (e.g., in private 5G [203, 204] or RAN slicing [164] scenarios). Specifically, RB scheduler can also take into account app context (e.g., video frame size, inference workload), which helps guarantee throughput at a more fine-grained level (e.g., per-frame latency guarantee). While Tutti [4] recently studied a similar idea, it only considers RB scheduling (not the end-to-end pipeline including DNN inference) as well as does not consider joint scheduling.

7.2.3 Impact of Hardware Evolution on Heimdall

7.2.3.1 Mobile GPU Evolution

Even when mobile GPUs evolve similar to desktop GPUs, the need for an app-aware coordination platform to dynamically schedule multiple tasks to satisfy the AR app requirements will persist.

Parallelization. With the architecture support, we can consider porting desktop GPU computing platforms (e.g., recent CUDA for ARM server platforms [205]) and spatially partitioning the GPU to run multi-DNN and rendering tasks concurrently. However, due to a limited number of computing cores and power of mobile GPUs (e.g., RTX 2080Ti: 13.45 TFLOPs vs. Adreno 640: 954 GFLOPs), static partitioning would be limited in running multiple compute-intensive DNNs. Instead, a coordinator should dynamically allocate resources at runtime; when

an inference request for a heavy DNN with high priority is enqueued, the co-ordinator should allocate more number of partitioned resources dynamically to minimize response time.

Preemption. With fine-grained, near-zero overhead preemption support (e.g., NVIDIA Pascal GPUs [206] support instruction-level preemption at 0.1 ms scale [207]), we can consider employing prior multi-DNN scheduling for desktop GPUs [69, 71]. However, prior works mostly assume that the task priorities are fixed in advance, whereas in AR apps they can be dynamic depending on the scene contents (e.g., in the surroundings monitoring scenario, face detection would need to run more frequently than object detection in case there are many people). Therefore, a coordinator would be needed to dynamically adjust priorities at runtime for app usability.

7.2.3.2 Emergence of NPUs/TPUs

Recently, neural processors are being prevalent in mobile/edge devices (e.g., Google Pixel edge TPU [208], Huawei Kirin NPU [209]). Several recent works also utilize such neural processors to run the multi-DNN inference workload (e.g., HERTI [210]-NPU, Band [211]-DSP and NPU). Such processors maximize computing power by packing a large number of cores specialized for DNN inference. For example, Google TPUs employ 128×128 systolic array-based matrix units (MXUs), which accelerate matrix multiplication by hard-wired calculation without memory access. We envision that Heimdall’s Pseudo-Preemption mechanism can also be useful in coordinating multiple tasks on such neural processors, as i) it is challenging to preempt the hard-wired MXUs, and ii) context switch overhead on bandwidth-limited mobile SoCs can be more costly due to larger state sizes than GPUs.

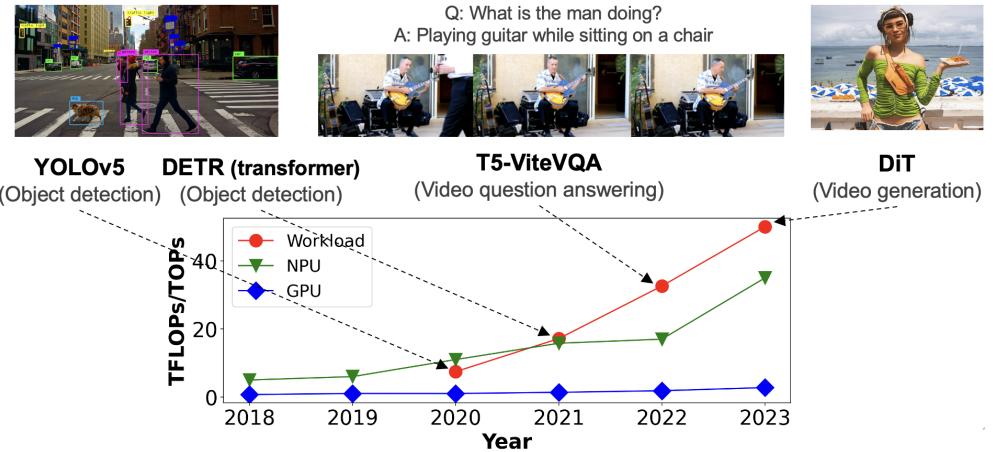


Figure 7.1: DNN inference complexity (assuming 1080p@30fps input) vs. hardware capability (NPU: Apple A12-A17, GPU: Qualcomm Adreno 630-740).

7.3 Future Projection: Would the Importance of System Optimization Persist?

With hardware evolution (e.g., NPU/TPU) as well as emergence of next-generation wireless networks (e.g., 5G/6G, Wi-Fi 6), production systems are provisioned more network and compute resources. While such trend alleviates the challenges in supporting the concurrent multi-DNN and rendering workload of emerging live video analytics apps, we believe that the importance of system optimization will continue, as the workload is growing at a faster speed than the resource capacity increase.

7.3.1 DNN complexity increase vs. hardware evolution

Emerging live video analytics apps require more number of complex analysis tasks for higher levels of video understanding and user immersiveness. For example, surveillance apps require spatio-temporal video analysis for complex event detection (e.g., action recognition, anomaly detection), and MR apps require NeRF for realistic 3D content generation. Furthermore, DNN architec-

tures are becoming more deeper and complex for higher accuracy. For example, vision transformers are outperforming traditional CNNs for various video analysis tasks (e.g., object detection) [212, 213], but at the cost of $10\text{--}30\times$ latency increase. Figure 7.1 shows that the inference complexity growth speed is much faster than the hardware compute capabilities. System optimization techniques including **EagleEye**'s content-aware adaptation and edge-cloud cooperative inference, **Heimdall**'s multi-DNN concurrency support will be more important to support such complex workloads on resource-constrained edge devices with low latency.

7.3.2 Video bitrate increase vs. network evolution

Furthermore, emerging live video analytics apps require high-resolution, multi-modal video processing. For example, city-scale surveillance requires multi-view 4K/8K and 360° video processing for wide coverage, and autonomous driving and AR require 3D point cloud video processing for accurate 3D perception. The data rate of such high-quality, multi-modal videos exceed the capacity of networks. For example, Figure 7.2 compares the video bitrate and network bandwidth (target per-user bandwidths and actual measurements). 3D point cloud video data rate is >1 Gbps [214]), whereas 5G eMBB (Enhanced Mobile Broadband) aims at 100 Mbps user-experienced throughput. Recent study also shows that practical 5G deployments only support ≈ 30 Mbps uplink throughput [4], resulting in long latency for high-resolution video streaming. Thus, content-aware adaptation techniques (e.g., adaptive face recognition in **EagleEye**, network-compute joint scheduling in **Pendulum**) will still be crucial to optimize data size and latency.

Furthermore, physical layer bandwidth increase in next-generation networks does not directly convert to the application latency gain. For example, when comparing the **EagleEye** offloading latency (Chapter 2.2.2) in 4G (11 Mbps)

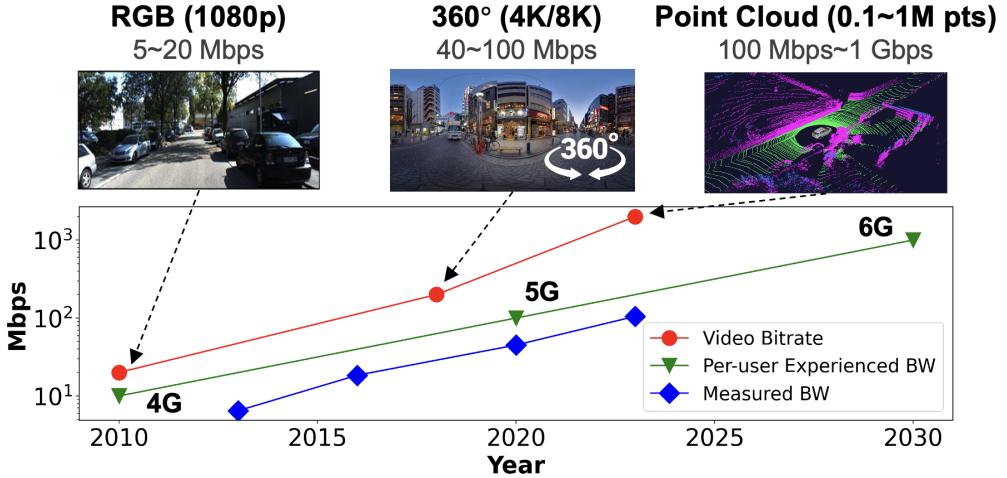


Figure 7.2: Video bitrate vs. network bandwidth (Green: target per-user experienced bandwidths of 4G, 5G, 6G, Blue: actual measurements from existing traces [3, 4] and our own measurements).

and 5G (45 Mbps), the end-to-end app latency gain (3.4 vs 2.1 seconds) does not proportionally increase with the $\approx 4\times$ bandwidth increase. This is due to several reasons. First, RAN resource scheduling affects the app-layer latency. For example, video traffic is classified as best effort in 5G QoS specification, and it suffers from queueing delay from long traffic flows (e.g., file transfer) [186]. Also, the buffer status report and uplink resource grant also has unpredictable delays, which can exceed 100 ms for commercial 5G [4]. Second, device power management strategy also affects the latency. For example, 4G LTE uses DRX mode, which makes UEs go idle states periodically to save energy, however waking up the idle mode for data transfer incurs about 30 ms latency [215].

7.4 Future Works

7.4.1 System Support for 3D Point Cloud Videos

Our current platform is mostly designed for high-resolution RGB videos. We plan to extend our platform support for more sensor modalities (e.g., RGB-D, LiDAR) for diverse live video analytics applications. As an initial attempt, we are designing an end-to-end live video analytics platform for 3D point cloud videos. 3D point cloud analysis add a new dimension of depth perception, which is crucial for various live video analytics apps (e.g., autonomous driving, indoor robot navigation). Especially, processing the 3D point cloud yields higher accuracy than 2D projection with lower number of FLOPs (e.g., 5% higher accuracy with $7 \times$ small MACs [216]). However, streaming and analyzing 3D point cloud is challenging due to its large data size. For example, our preliminary study shows that the end-to-end Octree-based streaming and 3D object detection pipeline takes ≈ 800 ms for 110 K points LiDAR frame. Fast and efficient 3D point cloud is also non-trivial, mainly due to its high data sparsity. Despite the challenges, we identified a key opportunity for optimization: objects only compose $\approx 10\%$ for typical 3D point cloud frames. We are currently developing a system that efficiently selects among multiple sampling features (e.g., edge filtering, resolution scaling, temporal tracking) to efficiently sample high-saliency points (and thus the streaming and analysis latency) from the input point cloud without accuracy drop.

7.4.2 App-RAN Cross-Layer Optimization

Pendulum currently takes the network-as-a-black-box approach (i.e., app-level bandwidth estimation and bitrate control). We plan to extend joint scheduling to app-RAN cross layer control. Specifically, we expect the following scheduling gains if the platform operator has control over both the RAN (e.g., private

5G [203, 204]) and the cloud server. *i) Better Scheduling Accuracy and Cost Reduction.* The RAN can provide a more accurate estimate of the network bandwidth to the cloud server based on the monitored channel status at the physical layer (e.g., uplink SINR) [171]. The RAN can also dynamically allocate bandwidth (Resource Blocks, RBs) across users considering their video content and network-compute tradeoffs (e.g., higher bandwidth to users with more dynamic scenes) to reduce overall cost. For example, our preliminary experiment with real-world video traces from BDD and MOT datasets shows that app-aware RB scheduling reduces the overall compute cost by up to 52% compared to app-agnostic equal RB scheduling. *ii) Per-Frame Latency Guarantee.* By jointly scheduling the RB transmission order as well as the GPU inference order, we can improve per-frame latency predictability and latency SLO satisfaction ratio. For example, in case a user’s frame transmission is unexpectedly delayed due to SINR fluctuation, we can prioritize his DNN inference at the cloud server to compensate for the delay and meet the latency deadline. We plan to integrate such cross-layer control in our future work.

Bibliography

- [1] L. Liu, H. Li, and M. Gruteser, “Edge assisted real-time object detection for mobile augmented reality,” in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–16.
- [2] J. Deng, J. Guo, Y. Zhou, J. Yu, I. Kotsia, and S. Zafeiriou, “RetinaFace: Single-stage dense face localisation in the wild,” *arXiv preprint arXiv:1905.00641*, 2019.
- [3] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, “Mahimahi: Accurate Record-and-Replay for HTTP,” in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, 2015, pp. 417–429.
- [4] D. Xu, A. Zhou, G. Wang, H. Zhang, X. Li, J. Pei, and H. Ma, “Tutti: coupling 5g ran and mobile edge computing for latency-critical video analytics,” in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, 2022, pp. 729–742.
- [5] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, “Live video analytics at scale with approximation and {Delay-Tolerance},” in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 377–392.

- [6] X. Liu, P. Ghosh, O. Ulutan, B. Manjunath, K. Chan, and R. Govindan, “Caesar: cross-camera complex activity recognition,” in *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, 2019, pp. 232–244.
- [7] J. Yi, S. Choi, and Y. Lee, “EagleEye: Wearable camera-based person identification in crowded urban spaces,” in *Proceedings of the 16th Annual International Conference on Mobile Computing and Networking*. ACM, 2020.
- [8] J. Yi and Y. Lee, “Heimdall: mobile gpu coordination platform for augmented reality applications,” in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–14.
- [9] Z. Li, M. Annett, K. Hinckley, K. Singh, and D. Wigdor, “HoloDoc: Enabling mixed reality workspaces that harness physical and digital content,” in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–14.
- [10] X. Zeng, B. Fang, H. Shen, and M. Zhang, “Distream: scaling live video analytics with workload-adaptive distributed edge intelligence,” in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 409–421.
- [11] H. Mao, R. Netravali, and M. Alizadeh, “Neural adaptive video streaming with pensieve,” in *Proceedings of the conference of the ACM special interest group on data communication*, 2017, pp. 197–210.
- [12] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, “A control-theoretic approach for dynamic adaptive video streaming over http,” in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 325–338.

- [13] J. Jiang, V. Sekar, and H. Zhang, “Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive,” in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, 2012, pp. 97–108.
- [14] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, “Spinn: synergistic progressive inference of neural networks over device and cloud,” in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–15.
- [15] “Samsung Galaxy S9 AR Emoji,” <https://www.sammobile.com/news/galaxy-s9-ar-emoji-explained-how-to-create-and-use-them/>. Accessed: 25 Mar. 2020.
- [16] A. Mathur, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar, “DeepEye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware,” in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2017, pp. 68–81.
- [17] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang, “EAST: an efficient and accurate scene text detector,” in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 5551–5560.
- [18] L. N. Huynh, Y. Lee, and R. K. Balan, “DeepMon: Mobile gpu-based deep learning framework for continuous vision applications,” in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2017, pp. 82–95.
- [19] X. Zeng, K. Cao, and M. Zhang, “MobileDeepPill: A small-footprint mobile deep learning system for recognizing unconstrained pill images,” in

Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services. ACM, 2017, pp. 56–67.

- [20] M. Xu, M. Zhu, Y. Liu, F. X. Lin, and X. Liu, “DeepCache: Principled cache for mobile deep vision,” in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. ACM, 2018, pp. 129–144.
- [21] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, “ArcFace: Additive angular margin loss for deep face recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4690–4699.
- [22] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, “Encoder-decoder with atrous separable convolution for semantic image segmentation,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 801–818.
- [23] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” in *IEEE CVPR*, 2017.
- [24] C. Zimmermann and T. Brox, “Learning to estimate 3d hand pose from single rgb images,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 4903–4911.
- [25] L. Engstrom, “Fast style transfer,” <https://github.com/lengstrom/fast-style-transfer/>, 2016.
- [26] “Microsoft HoloLens 2,” <https://www.microsoft.com/en-us/hololens/>. Accessed: 25 Mar. 2020.
- [27] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, “Convolutional pose machines,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4724–4732.

- [28] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “MobileNets: Efficient convolutional neural networks for mobile vision applications,” in *arXiv preprint arXiv:1704.04861*, 2017.
- [29] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [30] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.
- [31] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2117–2125.
- [32] X. Xie and K.-H. Kim, “Source compression with bounded dnn perception loss for iot edge computer vision,” in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–16.
- [33] “Ultralytics YOLOv5,” <https://github.com/ultralytics/yolov5>. Accessed: 1 Feb. 2024.
- [34] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler, “Motchallenge 2015: Towards a benchmark for multi-target tracking,” *arXiv preprint arXiv:1504.01942*, 2015.
- [35] “User Equipment (UE) radio access capabilities (3GPP TS 38.306 version 17.0.0 Release 17).” https://www.etsi.org/deliver/etsi_ts/138300_138399/138306/17.00.00_60/ts_138306v170000p.pdf. Accessed: 1 Feb. 2024.

- [36] “GSMA. 2020. 5G TDD Synchronisation Guidelines and Recommendations for the Coexistence of TDD Networks in the 3.5 GHz Range.” <https://www.gsma.com/spectrum/wp-content/uploads/2020/04/3.5-GHz-5G-TDD-Synchronisation.pdf>. Accessed: 1 Feb. 2024.
- [37] “TensorFlow-Lite on GPU for Mobile,” https://www.tensorflow.org/lite/performance/gpu_advanced. Accessed: 15 Dec. 2019.
- [38] “XiaoMi Mobile AI Compute Engine (MACE),” <https://github.com/XiaoMi/mace>. Accessed: 25 Mar. 2020.
- [39] P. Hu and D. Ramanan, “Finding tiny faces,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 951–959.
- [40] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, “Chameleon: scalable adaptation of video analytics,” in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 253–266.
- [41] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, “Towards wearable cognitive assistance,” in *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. ACM, 2014, pp. 68–81.
- [42] P. Jain, J. Manweiler, and R. Roy Choudhury, “OverLay: Practical mobile augmented reality,” in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2015, pp. 331–344.
- [43] K. Chen, T. Li, H.-S. Kim, D. E. Culler, and R. H. Katz, “MARVEL: Enabling mobile augmented reality with low energy and low latency,” in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2018, pp. 292–304.

- [44] A. Padmanabhan, N. Agarwal, A. Iyer, G. Ananthanarayanan, Y. Shu, N. Karianakis, G. H. Xu, and R. Netravali, “Gemel: Model merging for memory-efficient, real-time video analytics at the edge,” in *USENIX NSDI*, April 2023.
- [45] F. Cangialosi, N. Agarwal, V. Arun, S. Narayana, A. Sarwate, and R. Netravali, “Privid: Practical,{Privacy-Preserving} video analytics queries,” in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 209–228.
- [46] R. Bhardwaj, Z. Xia, G. Ananthanarayanan, J. Jiang, Y. Shu, N. Kari-anakis, K. Hsieh, P. Bahl, and I. Stoica, “Ekya: Continuous learning of video analytics models on edge compute servers,” in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 119–135.
- [47] K. Mehrdad, G. Ananthanarayanan, K. Hsieh, J. J. , R. N. , Y. Shu, M. Alizadeh, and V. Bahl, “Recl: Responsive resource-efficient continuous learning for video analytics,” in *USENIX NSDI*, April 2023.
- [48] “Alibaba Mobile Neural Network (MNN),” <https://github.com/alibaba/MNN>. Accessed: 25 Mar. 2020.
- [49] “Qualcomm Neural Processing SDK for AI,” <https://developer.qualcomm.com/software/qualcomm-neural-processing-sdk>. Accessed: 25 Mar. 2020.
- [50] S. Bhattacharya and N. D. Lane, “Sparsifying deep learning layers for constrained resource inference on wearables,” in *Proc. ACM SenSys*, 2016.
- [51] N. D. Lane, P. Georgiev, and L. Qendro, “DeepEar: robust smartphone audio sensing in unconstrained acoustic environments using deep learn-

- ing,” in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2015, pp. 283–294.
- [52] S. Yao, Y. Zhao, H. Shao, S. Liu, D. Liu, L. Su, and T. Abdelzaher, “FastDeepIoT: Towards understanding and optimizing neural network execution time on mobile and embedded devices,” in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2018, pp. 278–291.
- [53] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, “On-demand deep model compression for mobile devices: A usage-driven model selection framework,” in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2018, pp. 389–400.
- [54] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, “DeepX: A software accelerator for low-power deep learning inference on mobile devices,” in *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*. IEEE Press, 2016, p. 23.
- [55] R. Lee, S. I. Venieris, L. Dudziak, S. Bhattacharya, and N. D. Lane, “MobiSR: Efficient on-device super-resolution through heterogeneous mobile processors,” in *The 25th Annual International Conference on Mobile Computing and Networking*. ACM, 2019, p. 54.
- [56] B. Fang, X. Zeng, and M. Zhang, “NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision,” in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. ACM, 2018, pp. 115–127.
- [57] S. Lee and S. Nirjon, “Fast and scalable in-memory deep multitask learning via neural weight virtualization,” in *Proceedings of the 18th Interna-*

- tional Conference on Mobile Systems, Applications, and Services*, 2020, pp. 175–190.
- [58] A. H. Jiang, D. L.-K. Wong, C. Canel, L. Tang, I. Misra, M. Kaminsky, M. A. Kozuch, P. Pillai, D. G. Andersen, and G. R. Ganger, “Mainstream: Dynamic stem-sharing for multi-tenant video processing,” in *2018 USENIX Annual Technical Conference (USENIX ATC)*, 2018, pp. 29–42.
- [59] I. Tanasic, I. Gelado, J. Cabezas, A. Ramirez, N. Navarro, and M. Valero, “Enabling preemptive multiprogramming on GPUs,” in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE, 2014, pp. 193–204.
- [60] J. J. K. Park, Y. Park, and S. Mahlke, “Chimera: Collaborative preemption for multitasking on a shared GPU,” *ACM SIGPLAN Notices*, vol. 50, no. 4, pp. 593–606, 2015.
- [61] Z. Wang, J. Yang, R. Melhem, B. Childers, Y. Zhang, and M. Guo, “Simultaneous multikernel GPU: Multi-tasking throughput processors via fine-grained sharing,” in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 358–369.
- [62] “NVIDIA Hyper-Q,” http://developer.download.nvidia.com/compute/DevZone/C/html_x64/6_Advanced/simpleHyperQ/doc/HyperQ.pdf. Accessed: 25 Mar. 2020.
- [63] “NVIDIA GPU virtualization,” <https://www.nvidia.com/ko-kr/data-center/graphics-cards-for-virtualization/>. Accessed: 25 Mar. 2020.
- [64] S. Pai, M. J. Thazhuthaveetil, and R. Govindarajan, “Improving GPGPU concurrency with elastic kernels,” in *ACM SIGPLAN Notices*, vol. 48, no. 4. ACM, 2013, pp. 407–418.

- [65] M. Lee, S. Song, J. Moon, J. Kim, W. Seo, Y. Cho, and S. Ryu, “Improving GPGPU resource utilization through alternative thread block scheduling,” in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2014, pp. 260–271.
- [66] B. Wu, G. Chen, D. Li, X. Shen, and J. Vetter, “Enabling and exploiting flexible task assignment on GPU through SM-centric program transformations,” in *Proceedings of the 29th ACM on International Conference on Supercomputing*. ACM, 2015, pp. 119–130.
- [67] R. Ausavarungnirun, V. Miller, J. Landgraf, S. Ghose, J. Gandhi, A. Jog, C. J. Rossbach, and O. Mutlu, “MASK: Redesigning the GPU memory hierarchy to support multi-application concurrency,” in *ACM SIGPLAN Notices*, vol. 53, no. 2. ACM, 2018, pp. 503–518.
- [68] Z. Fang, D. Hong, and R. K. Gupta, “Serving deep neural networks at the cloud edge for vision applications on mobile platforms,” in *Proceedings of the 10th ACM Multimedia Systems Conference*, 2019, pp. 36–47.
- [69] H. Zhou, S. Bateni, and C. Liu, “S³DNN: Supervised streaming and scheduling for GPU-accelerated real-time DNN workloads,” in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 190–201.
- [70] M. Yang, S. Wang, J. Bakita, T. Vu, F. D. Smith, J. H. Anderson, and J.-M. Frahm, “Re-thinking CNN frameworks for time-sensitive autonomous-driving applications: Addressing an industrial challenge,” in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 305–317.
- [71] Y. Xiang and H. Kim, “Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference,” in *IEEE RTSS*, 2019.

- [72] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, “MCDNN: An approximation-based execution framework for deep stream processing under resource constraints,” in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2016, pp. 123–136.
- [73] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, “DeepDecision: A mobile deep learning framework for edge video analytics,” in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1421–1429.
- [74] P. Jain, J. Manweiler, and R. Roy Choudhury, “Low bandwidth offload for mobile ar,” in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. ACM, 2016, pp. 237–251.
- [75] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, “Glimpse: Continuous, real-time object recognition on mobile devices,” in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2015, pp. 155–168.
- [76] L. Liu, H. Li, and M. Gruteser, “Edge assisted real-time object detection for mobile augmented reality,” in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. ACM, 2019.
- [77] J. Hu, A. Shearer, S. Rajagopalan, and R. LiKamWa, “Banner: An image sensor reconfiguration framework for seamless resolution-based tradeoffs,” in *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2019, pp. 236–248.
- [78] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. Netravali, “Reducto: On-camera filtering for resource-efficient real-time video ana-

- lytics,” in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 359–376.
- [79] K. Hsieh, G. Ananthanarayanan, P. Bodik, S. Venkataraman, P. Bahl, M. Philipose, P. B. Gibbons, and O. Mutlu, “Focus: Querying large video datasets with low latency and low cost,” in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 269–286.
- [80] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, “Noscope: optimizing neural network queries over video at scale,” *arXiv preprint arXiv:1703.02529*, 2017.
- [81] C. Canel, T. Kim, G. Zhou, C. Li, H. Lim, D. G. Andersen, M. Kaminsky, and S. Dulloor, “Scaling video analytics on constrained edge nodes,” *Proceedings of Machine Learning and Systems*, vol. 1, pp. 406–417, 2019.
- [82] T. Zhang, A. Chowdhery, P. Bahl, K. Jamieson, and S. Banerjee, “The design and implementation of a wireless video surveillance system,” in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, 2015, pp. 426–438.
- [83] C. Pakha, A. Chowdhery, and J. Jiang, “Reinventing video streaming for distributed vision analytics,” in *10th USENIX workshop on hot topics in cloud computing (HotCloud 18)*, 2018.
- [84] M. Xu, T. Xu, Y. Liu, and F. X. Lin, “Video analytics with zero-streaming cameras,” in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 459–472.

- [85] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, “Aw-stream: Adaptive wide-area streaming analytics,” in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 236–252.
- [86] W. Zhang, Z. He, L. Liu, Z. Jia, Y. Liu, M. Gruteser, D. Raychaudhuri, and Y. Zhang, “Elf: accelerate high-resolution mobile deep vision with content-aware parallel offloading,” in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, 2021, pp. 201–214.
- [87] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, “Server-driven video streaming for deep learning inference,” in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 557–570.
- [88] Y. Wang, W. Wang, J. Zhang, J. Jiang, and K. Chen, “Bridging the edge-cloud barrier for real-time advanced vision analytics,” in *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019.
- [89] H. Yeo, Y. Jung, J. Kim, J. Shin, and D. Han, “Neural adaptive content-aware internet video delivery,” in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 645–661.
- [90] J. Kim, Y. Jung, H. Yeo, J. Ye, and D. Han, “Neural-enhanced live streaming: Improving live video ingest via online learning,” in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 107–125.

- [91] K. Du, Q. Zhang, A. Arapin, H. Wang, Z. Xia, and J. Jiang, “Accmpeg: Optimizing video encoding for accurate video analytics,” *Proceedings of Machine Learning and Systems*, vol. 4, pp. 450–466, 2022.
- [92] F. Romero, M. Zhao, N. J. Yadwadkar, and C. Kozyrakis, “Llama: A heterogeneous & serverless framework for auto-tuning video analytics pipelines,” in *Proceedings of the ACM Symposium on Cloud Computing*, 2021, pp. 1–17.
- [93] H. Shen, L. Chen, Y. Jin, L. Zhao, B. Kong, M. Philipose, A. Krishnamurthy, and R. Sundaram, “Nexus: A gpu cluster engine for accelerating dnn-based video analysis,” in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 322–337.
- [94] R. LiKamWa and L. Zhong, “Starfish: Efficient concurrency support for computer vision applications,” in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2015, pp. 213–226.
- [95] Y. Lee, A. Scolari, B.-G. Chun, M. D. Santambrogio, M. Weimer, and M. Interlandi, “pretzel: Opening the black box of machine learning prediction serving systems,” in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 611–626.
- [96] H. Guo, S. Yao, Z. Yang, Q. Zhou, and K. Nahrstedt, “Crossroi: cross-camera region of interest optimization for efficient real time video analytics at scale,” in *Proceedings of the 12th ACM Multimedia Systems Conference*, 2021, pp. 186–199.
- [97] S. Jain, X. Zhang, Y. Zhou, G. Ananthanarayanan, J. Jiang, Y. Shu, P. Bahl, and J. Gonzalez, “Spatula: Efficient cross-camera video analyt-

- ics on large camera networks,” in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2020, pp. 110–124.
- [98] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, “Clipper: A low-latency online prediction serving system,” in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 613–627.
- [99] J. Yi, S. Kim, J. Kim, and S. Choi, “Supremo: Cloud-assisted low-latency super-resolution in mobile devices,” *IEEE Transactions on Mobile Computing*, 2020.
- [100] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, “Neurosurgeon: Collaborative intelligence between the cloud and mobile edge,” *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.
- [101] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and T. Abdelzaher, “Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency,” in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 476–488.
- [102] L. Yu and W. Xiang, “X-pruner: explainable pruning for vision transformers,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 24355–24363.
- [103] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, “Smoothquant: Accurate and efficient post-training quantization for large language models,” in *International Conference on Machine Learning*. PMLR, 2023, pp. 38087–38099.

- [104] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “Squeezezenet: Alexnet-level accuracy with 50x fewer parameters and 0.5 mb model size,” *arXiv preprint arXiv:1602.07360*, 2016.
- [105] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.
- [106] P. Guo, B. Hu, and W. Hu, “Mistify: Automating {DNN} model porting for {On-Device} inference at the edge,” in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 705–719.
- [107] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, “Searching for mobilenetv3,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1314–1324.
- [108] M. Tan, R. Pang, and Q. V. Le, “Efficientdet: Scalable and efficient object detection,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 781–10 790.
- [109] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, “SphereFace: Deep hypersphere embedding for face recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 212–220.
- [110] H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, and W. Liu, “CosFace: Large margin cosine loss for deep face recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5265–5274.

- [111] J. Han and B. Bhanu, “Individual recognition using gait energy image,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 2, pp. 316–322, 2005.
- [112] H. Wang, X. Bao, R. Roy Choudhury, and S. Nelakuditi, “Visually fingerprinting humans without face recognition,” in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2015, pp. 345–358.
- [113] J. Chauhan, Y. Hu, S. Seneviratne, A. Misra, A. Seneviratne, and Y. Lee, “BreathPrint: Breathing acoustics-based user authentication,” in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2017, pp. 278–291.
- [114] K. R. Farrell, R. J. Mammone, and K. T. Assaleh, “Speaker recognition using neural networks and conventional classifiers,” *IEEE Transactions on speech and audio processing*, vol. 2, no. 1, pp. 194–205, 1994.
- [115] Y. Zhao, S. Wu, L. Reynolds, and S. Azenkot, “A face recognition application for people with visual impairments: Understanding use beyond the lab,” in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 2018, p. 215.
- [116] S. Panchanathan, S. Chakraborty, and T. McDaniel, “Social interaction assistant: a person-centered approach to enrich social interactions for individuals with visual impairments,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 5, pp. 942–951, 2016.
- [117] L. B. Neto, F. Grijalva, V. R. M. L. Maike, L. C. Martini, D. Florencio, M. C. C. Baranauskas, A. Rocha, and S. Goldenstein, “A kinect-based wearable face recognition system to aid visually impaired users,” *IEEE Transactions on Human-Machine Systems*, vol. 47, no. 1, pp. 52–64, 2016.

- [118] L. He, H. Li, Q. Zhang, and Z. Sun, “Dynamic feature learning for partial face recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7054–7063.
- [119] J. Lezama, Q. Qiu, and G. Sapiro, “Not afraid of the dark: NIR-VIS face recognition via cross-spectral hallucination and low-rank embedding,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6628–6637.
- [120] “TensorFlow-Lite Object Detection Demo,” https://www.tensorflow.org/lite/models/object_detection/overview. 15 Dec. 2019.
- [121] P. Li, L. Prieto, D. Mery, and P. J. Flynn, “On low-resolution face recognition in the wild: Comparisons and new techniques,” *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 8, pp. 2000–2012, 2019.
- [122] M. B. Lewis and A. J. Edmonds, “Face detection: Mapping human performance,” *Perception*, vol. 32, no. 8, pp. 903–920, 2003.
- [123] M. Kampf, I. Nachson, and H. Babkoff, “A serial test of the laterality of familiar face recognition,” *Brain and cognition*, vol. 50, no. 1, pp. 35–50, 2002.
- [124] Y. Guo, L. Zhang, Y. Hu, X. He, and J. Gao, “MS-Celeb-1M: A dataset and benchmark for large-scale face recognition,” in *European Conference on Computer Vision*. Springer, 2016, pp. 87–102.
- [125] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman, “VGGFace2: A dataset for recognising faces across pose and age,” in *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*. IEEE, 2018, pp. 67–74.

- [126] X. Tang, D. K. Du, Z. He, and J. Liu, “Pyramidbox: A context-assisted single shot face detector,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 797–813.
- [127] “EyeSight Rapter AR Glass,” <https://every sight.com/about-raptor/>. Accessed: 15 Dec. 2019.
- [128] M. Najibi, P. Samangouei, R. Chellappa, and L. S. Davis, “SSH: Single stage headless face detector,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 4875–4884.
- [129] Y. Chen, Y. Tai, X. Liu, C. Shen, and J. Yang, “FSRNet: End-to-end learning face super-resolution with facial priors,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2492–2501.
- [130] R. Zhang, “Making convolutional networks shift-invariant again,” *International Conference on Machine Learning (ICML)*, 2019.
- [131] B. Lim, S. Son, H. Kim, S. Nah, and K. Mu Lee, “Enhanced deep residual networks for single image super-resolution,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 136–144.
- [132] N. Ahn, B. Kang, and K.-A. Sohn, “Fast, accurate, and lightweight super-resolution with cascading residual network,” in *Proc. ECCV*, 2018.
- [133] A. Bulat and G. Tzimiropoulos, “Super-FAN: Integrated facial landmark localization and super-resolution of real-world low resolution faces in arbitrary poses with gans,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 109–117.

- [134] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, 2014, pp. 2672–2680.
- [135] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, “Improved training of wasserstein gans,” in *Advances in Neural Information Processing Systems*, 2017, pp. 5767–5777.
- [136] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*. IEEE, 2009, pp. 248–255.
- [137] J. Canny, “A computational approach to edge detection,” in *Readings in computer vision*. Elsevier, 1987, pp. 184–203.
- [138] S. Chen, Y. Liu, X. Gao, and Z. Han, “MobileFaceNets: Efficient CNNs for accurate real-time face verification on mobile devices,” in *Chinese Conference on Biometric Recognition*. Springer, 2018, pp. 428–438.
- [139] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, “Labeled faces in the wild: A database for studying face recognition in unconstrained environments,” University of Massachusetts, Amherst, Tech. Rep. 07-49, October 2007.
- [140] M. Yuan, L. Zhang, F. He, X. Tong, and X.-Y. Li, “Infi: end-to-end learnable input filter for resource-efficient mobile-centric inference,” in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, 2022, pp. 228–241.
- [141] S. K. Lam, A. Pitrou, and S. Seibert, “Numba: A llvm-based python jit compiler,” in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. ACM, 2015, p. 7.

- [142] S. Yang, P. Luo, C.-C. Loy, and X. Tang, “WIDER FACE: A face detection benchmark,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5525–5533.
- [143] T. Karras, S. Laine, and T. Aila, “A style-based generator architecture for generative adversarial networks,” *arXiv preprint arXiv:1812.04948*, 2018.
- [144] A. Bulat and G. Tzimiropoulos, “How far are we from solving the 2d & 3d face alignment problem?(and a dataset of 230,000 3d facial landmarks),” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1021–1030.
- [145] J. Guo and H. Chao, “One-to-many network for visually pleasing compression artifacts reduction,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3038–3047.
- [146] G. Lu, W. Ouyang, D. Xu, X. Zhang, Z. Gao, and M.-T. Sun, “Deep kalman filtering network for video compression artifact reduction,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 568–584.
- [147] “Microsoft and AT&T demonstrate 5G-powered video analytics,” <https://azure.microsoft.com/en-us/blog/microsoft-and-att-demonstrate-5gpowered-video-analytics/>. Accessed: 1 Feb. 2024.
- [148] M. Tan and Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.
- [149] “Computing Receptive Fields of Convolutional Neural Networks,” <https://distill.pub/2019/computing-receptive-fields/>. Accessed: 1 Feb. 2024.

- [150] W. Luo, Y. Li, R. Urtasun, and R. Zemel, “Understanding the effective receptive field in deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 29, 2016.
- [151] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, “Analysis and design of the google congestion control for web real-time communication (webrtc),” in *Proceedings of the 7th International Conference on Multimedia Systems*, 2016, pp. 1–12.
- [152] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, “Openairinterface: A flexible platform for 5g research,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 33–38, 2014.
- [153] “Sysmocom. 2022. Programmable SIM cards from Sysmocom.” <https://shop.sysmocom.de/sysmoISIM-SJA2-SIM-USIM-ISIM-Card-10-pack-with-ADM-keys/sysmoISIM-SJA2-10p-adm>. Accessed: 1 Feb. 2024.
- [154] “Linux traffic control,” <https://man7.org/linux/man-pages/man8/tc.8.html>. Accessed: 1 Feb. 2024.
- [155] “EC2 on-demand vs. reserved instance pricing.” https://aws.amazon.com/compare/the-difference-between-on-demand-instances-and-reserved-instances/?nc1=h_ls. Accessed: 1 Feb. 2024.
- [156] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, “Videoedge: Processing camera streams using hierarchical clusters,” in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 115–131.

- [157] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell, “Bdd100k: A diverse driving video database with scalable annotation tooling,” *arXiv preprint arXiv:1805.04687*, vol. 2, no. 5, p. 6, 2018.
- [158] “AT&T Cell Phone Plans,” <https://www.att.com/5g/consumer/>. Accessed: 1 Feb. 2024.
- [159] “T-Mobile Cell Phone Plans,” <https://www.t-mobile.com/cell-phone-plans>. Accessed: 1 Feb. 2024.
- [160] “Google Cloud Pricing,” <https://cloud.google.com/compute/gpus-pricing>. Accessed: 1 Feb. 2024.
- [161] “Amazon EC2 Pricing,” <https://aws.amazon.com/ko/ec2/pricing/on-demand/>. Accessed: 1 Feb. 2024.
- [162] J. Pont-Tuset, F. Perazzi, S. Caelles, P. Arbeláez, A. Sorkine-Hornung, and L. Van Gool, “The 2017 davis challenge on video object segmentation,” *arXiv preprint arXiv:1704.00675*, 2017.
- [163] “A Unified Architecture for Instance and Semantic Segmentation,” <http://presentations.cocodataset.org/COCO17-Stuff-FAIR.pdf>. Accessed: 1 Feb. 2024.
- [164] Y. Chen, R. Yao, H. Hassanieh, and R. Mittal, “Channel-aware 5g RAN slicing with customizable schedulers,” in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 1767–1782.
- [165] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International journal of computer vision*, vol. 60, pp. 91–110, 2004.
- [166] H. Bay, T. Tuytelaars, and L. Van Gool, “Surf: Speeded up robust features,” in *Computer Vision-ECCV 2006: 9th European Conference on*

Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9.
Springer, 2006, pp. 404–417.

- [167] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, “Serving DNNs like clockwork: Performance predictability from the bottom up,” in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 443–462.
- [168] A. Kirillov, K. He, R. Girshick, and P. Dollár, “A unified architecture for instance and semantic segmentation,” 2017.
- [169] X. Xie, X. Zhang, S. Kumar, and L. E. Li, “pistream: Physical layer informed adaptive video streaming over lte,” in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, 2015, pp. 413–425.
- [170] X. Xie, X. Zhang, and S. Zhu, “Accelerating mobile web loading using cellular link information,” in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, 2017, pp. 427–439.
- [171] Y. Xie, F. Yi, and K. Jamieson, “Pbe-cc: Congestion control via endpoint-centric, physical-layer bandwidth measurements,” in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 451–464.
- [172] “Open-RAN Alliance,” <https://www.o-ran.org/>. Accessed: 1 Feb. 2024.
- [173] W.-H. Ko, U. Dinesha, U. Ghosh, S. Shakkottai, D. Bharadia, and R. Wu, “EdgeRIC: Empowering realtime intelligent optimization and control in nextg networks,” *arXiv preprint arXiv:2304.11199*, 2023.

- [174] R. Schmidt, M. Irazabal, and N. Nikaein, “Flexric: an sdk for next-generation sd-rans,” in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, 2021, pp. 411–425.
- [175] “Architecture & E2 General Aspects and Principles. Technical Specification O-RAN.WG3.E2GAP-v01.01. O-RAN Working Group 3.”
- [176] “O-RAN Working Group 3, “O-RAN Near-Real-time RAN Intelligent Controller E2 Service Model (E2SM) KPM 4.0,” ORAN-WG3.E2SM-KPM-v04.00 Technical Specification, October 2023.”
- [177] X. Foukas, B. Radunovic, M. Balkwill, Z. Lai, and C. Settle, “Programmable ran platform for flexible real-time control and telemetry,” in *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, 2023, pp. 1–3.
- [178] “Microsoft programmable RAN platform with dynamic service models,” <https://azure.microsoft.com/en-us/resources/research/microsoft-programmable-ran-platform-with-dynamic-service-models>. Accessed: 1 Feb. 2024.
- [179] T.-W. Chin, R. Ding, and D. Marculescu, “Adascale: Towards real-time video object detection using adaptive scaling,” *Proceedings of machine learning and systems*, vol. 1, pp. 431–441, 2019.
- [180] A. Chaurasia and E. Culurciello, “Linknet: Exploiting encoder representations for efficient semantic segmentation,” in *2017 IEEE Visual Communications and Image Processing (VCIP)*. IEEE, 2017, pp. 1–4.
- [181] “FFMPEG,” <https://ffmpeg.org/>. Accessed: 1 Feb. 2024.

- [182] “Secure Reliable Transport (SRT) Protocol,” <https://github.com/Haivision/srt>. Accessed: 1 Feb. 2024.
- [183] “CppFlow,” <https://github.com/serizba/cppflow>. Accessed: 1 Feb. 2024.
- [184] “PyTorch,” <https://pytorch.org/>. Accessed: 1 Feb. 2024.
- [185] “OpenCV,” <https://opencv.org/>. Accessed: 1 Feb. 2024.
- [186] J. Kim, Y. Lee, H. Lim, Y. Jung, S. M. Kim, and D. Han, “Outran: co-optimizing for flow completion time in radio access network,” in *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies*, 2022, pp. 369–385.
- [187] “OAI-5G RF Simulator,” https://gitlab.eurecom.fr/oai/openairinterface5g/-/tree/develop/radio/rfsimulator?ref_type=heads. Accessed: 1 Feb. 2024.
- [188] “Google Cloud Serverless Computing,” <https://cloud.google.com/serverless?hl=en>. Accessed: 1 Feb. 2024.
- [189] “AWS Lambda Serverless Computing,” <https://aws.amazon.com/ko/lambda/pricing/>. Accessed: 1 Feb. 2024.
- [190] K. Zhang, B. He, J. Hu, Z. Wang, B. Hua, J. Meng, and L. Yang, “G-NET: Effective GPU sharing in NFV systems,” in *15th USENIX Symposium on Networked Systems Design and Implementation NSDI 18)*, 2018, pp. 187–200.
- [191] G. Wang, Y. Lin, and W. Yi, “Kernel fusion: An effective method for better power efficiency on multithreaded gpu,” in *2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*. IEEE, 2010, pp. 344–350.

- [192] J. Lee, N. Chirkov, E. Ignasheva, Y. Pisarchyk, M. Shieh, F. Riccardi, R. Sarokin, A. Kulik, and M. Grundmann, “On-device neural net inference with mobile gpus,” *arXiv preprint arXiv:1907.01989*, 2019.
- [193] “NVIDIA’s next generation CUDA compute architecture: Kepler GK110, 2012.” <https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf>. Accessed: 25 Mar. 2020.
- [194] M. Guevara, C. Gregg, K. Hazelwood, and K. Skadron, “Enabling task parallelism in the CUDA scheduler,” in *Workshop on Programming Models for Emerging Architectures*, vol. 9. Citeseer, 2009.
- [195] “Snapdragon 845: Immersing you in a brave new world of XR.” <https://www.qualcomm.com/news/onq/2018/01/18/snapdragon-845-immersing-you-brave-new-world-xr>. Accessed: 25 Mar. 2020.
- [196] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.
- [197] C. Hwang, S. Pushp, C. Koh, J. Yoon, Y. Liu, S. Choi, and J. Song, “RAVEN: Perception-aware optimization of power consumption for mobile games,” in *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*. ACM, 2017, pp. 422–434.
- [198] “XiaoMi Mobile AI Compute Engine (MACE) model zoo,” <https://github.com/XiaoMi/mace-models>. Accessed: 25 Mar. 2020.
- [199] S. Zafeiriou, G. Tzimiropoulos, and M. Pantic, “The 300 videos in the wild (300-VW) facial landmark tracking in-the-wild challenge,” in *ICCV Workshop*, vol. 32, 2015, p. 73.

- [200] “Qualcomm Snapdragon Profiler,” <https://developer.qualcomm.com/software/snapdragon-profiler>. Accessed: 25 Mar. 2020.
- [201] T. Jin, S. He, and Y. Liu, “Towards accurate GPU power modeling for smartphones,” in *Proceedings of the 2nd Workshop on Mobile Gaming*, 2015, pp. 7–11.
- [202] J. J. Kang and S. Adibi, “Bushfire disaster monitoring system using low power wide area networks (lpwan),” *Technologies*, vol. 5, no. 4, p. 65, 2017.
- [203] A. Ajaz, “Private 5g: The future of industrial wireless,” *IEEE Industrial Electronics Magazine*, vol. 14, no. 4, pp. 136–145, 2020.
- [204] M. Wen, Q. Li, K. J. Kim, D. López-Pérez, O. A. Dobre, H. V. Poor, P. Popovski, and T. A. Tsiftsis, “Private 5g networks: concepts, architectures, and research landscape,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 16, no. 1, pp. 7–25, 2021.
- [205] “NVIDIA CUDA on Arm,” <https://developer.nvidia.com/cuda-toolkit/arm>. Accessed: 25 Mar. 2020.
- [206] “NVIDIA Tesla P100, 2016.” <https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf>. Accessed: 25 Mar. 2020.
- [207] “R. Smith and Anandtech. Preemption improved: Fine-grained pre-emption for time-critical tasks, 2016.” <http://www.anandtech.com/show/10325/the-nvidia-geforce-gtx-1080-and-1070-founders-edition-review/10>. Accessed: 25 Mar. 2020.
- [208] “Google Edge TPU,” <https://cloud.google.com/edge-tpu?hl=en>. Accessed: 25 Mar. 2020.

- [209] “Huawei Kirin SoC with NPU,” <http://www.hisilicon.com/en/Products/ProductList/Kirin>. Accessed: 25 Mar. 2020.
- [210] M. Han and W. Baek, “Herti: A reinforcement learning-augmented system for efficient real-time inference on heterogeneous embedded systems,” in *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2021, pp. 90–102.
- [211] J. S. Jeong, J. Lee, D. Kim, C. Jeon, C. Jeong, Y. Lee, and B.-G. Chun, “Band: coordinated multi-dnn inference on heterogeneous mobile processors,” in *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, 2022, pp. 235–247.
- [212] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid, “Vivit: A video vision transformer,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 6836–6846.
- [213] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” *arXiv preprint arXiv:2010.11929*, 2020.
- [214] K. Lee, J. Yi, Y. Lee, S. Choi, and Y. M. Kim, “Groot: a real-time streaming system of high-fidelity volumetric videos,” in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–14.
- [215] Z. Tan, J. Zhao, Y. Li, Y. Xu, and S. Lu, “{Device-Based}{LTE} latency reduction at the application layer,” in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 471–486.

- [216] Y. Lin, Z. Zhang, H. Tang, H. Wang, and S. Han, “Pointacc: Efficient point cloud accelerator,” in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 449–461.

초 록

실시간 비디오 분석은 교통 모니터링, 감시, 개인 식별 및 AR/MR 등 다양한 유용한 서비스의 핵심이 되는 기술이다. 하지만, 강건하고 효율적인 실시간 비디오 분석 시스템을 디자인 하는데에는 많은 어려움이 따른다. 핵심 문제는 자원 한정적인 모바일 기기에서 비디오 스트림을 실시간으로 정밀히 분석하고, 분석 결과를 사용자에게 전달 및 상호 작용을 가능하게 하는 것이다. 특히, 이를 위해서는 고해상도 비디오에 다수의 심층 신경망 (DNNs)을 연속적으로, 동시에 실행해야 한다. 본 논문에서는 MR, 자율주행 등 미래형 실시간 비디오 분석 응용의 워크로드를 특징짓고, 해당 워크로드를 지원하기 위한 엣지-클라우드 협력적 플랫폼을 설계한다. 구체적으로 엣지, 네트워크, 클라우드를 아우르는 엔드-투-엔드 최적화를 수행하여 작업을 실시간 처리량, 낮은 프레임 지연 시간 및 높은 정확도 성능을 지원한다.

먼저, 본 논문에서는 혼잡한 도시 공간에서 대상 사람(예: 실종 아동, 도주 중인 범인)을 실시간으로 추적하는 AR 시스템 EagleEye를 디자인한다. 얼굴 식별은 매 고해상도 비디오 프레임마다 복잡한 DNN의 시퀀스의 반복적 수행을 요구하여, 자원 한정적인 모바일 기기에서 실시간 수행이 매우 어렵다. EagleEye의 핵심 기술은 콘텐츠 적응형 병렬 실행으로, 얼굴 해상도, 자세 등 인식 난이도에 따라 다중 DNN 얼굴 식별 파이프라인을 적응적으로 조절하고, 이를 모바일 및 클라우드의 이기종 프로세서를 협력적으로 활용하여 실시간 수행하는 기술이다. 또한, 본 논문에서는 대상의 예시 얼굴 이미지를 활용하여 저해상도로 캡처된 얼굴에서 얼굴의 세부 정보를 복원하여 정확한 인식을 가능하게 하는 새로운 ICN 및 그 학습 방법론을 디자인한다. 다양한 실환경 성능 평가 결과, ICN이 저해상도 얼굴

인식 정확도를 크게 향상시키며, 매 1080p 비디오 프레임 당 108 kB의 데이터를 전송만으로 프레임 지연 시간을 최대 9.07배 가속화하는 것을 검증하였다.

다음으로, 본 논문에서는 네트워크-컴퓨트 공동 스케줄링을 통한 앤드-투-엔드 실시간 비디오 분석 시스템 Pendulum을 디자인한다. 비디오 분석 시스템은 동적인 비디오 콘텐츠와 사용 자원 변동으로 인해 네트워크(비디오 스트리밍) 및 컴퓨터(DNN 추론) 단계에서의 자원 병목이 복잡한 패턴으로 번갈아 발생한다. 하지만, 기존 시스템은 네트워크 또는 컴퓨터 단일 스케줄링에 한정되어, 지연 시간/정확도 성능 저하 및 리소스 낭비 문제를 겪는다. 이러한 한계를 극복하기 위해, 본 논문에서는 비디오 비트레이트와 DNN 모델 크기 간의 트레이드오프 관계를 새롭게 발견한다. 이를 활용하여, (i) 효율적이고 확장 가능한 네트워크-컴퓨트 공동 스케줄링 메커니즘, (ii) 경량 트레이드오프 프로파일러 및 (iii) 다중 사용자 공동 자원 스케줄러로 구성된 앤드-투-엔드 시스템을 디자인한다. 다양한 데이터셋 및 최신 DNN에 대한 실험 결과, Pendulum이 최신 단일 스케줄링 시스템 대비 최대 0.64 mIoU 향상 및 1.29배 높은 처리량을 달성하는 것을 검증하였다.

마지막으로, 본 논문에서는 모바일 GPU에서 다중 DNN 및 렌더링 동시 수행을 지원하는 모바일 플랫폼 Heimdall을 디자인한다. 기존 모바일 딥러닝 프레임워크는 자원경쟁이 없는 환경에서 단일 DNN 실행을 가정하고 설계되어, 다중 DNN 및 렌더링 동시 수행 시 심각한 성능 저하를 겪는다(예: 추론 지연이 59.93에서 1181 ms로 증가, 렌더링 프레임 속도가 30에서 12 fps로 감소). 다중 작업 스케줄링은 데스크톱 GPU에 대해 활발히 연구되었지만, 모바일 GPU에서는 제한된 아키텍처 지원 및 메모리 대역폭으로 인해 적용이 어렵다. 이를 해결하기 위해, 우리는 유사 선점 메커니즘을 디자인하여 DNN을 렌더링 지연 시간을 고려하여 작은 단위로 분할하고, 동시에 수행되는 GPU 수행작업 간 우선순위를 지정하고 응용 요구사항을 고려하여 동적으로 스케줄링한다. 다양한 MR 응용 시나리오에 대한 성능 평가 결과, Heimdall은 DNN 추론 지연 시간을 기존의 멀티 스레딩 접근에 비해 약 15 배 감소시키며, 프레임 속도를 11.99에서 29.96 fps로 향상시키는 것을 검증하였다.

주요어: 실시간 비디오 분석, 엣지-클라우드 협력적 시스템, 모바일/엣지 AI

학번: 2020-39481