上海交通大学

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

BACHELOR'S THESIS

论文题目：Cross Camera Video Analytics at Edge

学生姓名：岑盛、金哲健、杨煦庭、郑智睿、潘启颖

学生学号：517370910045, 517370910167, 517370910013, 517370910044, 517370910039

专　　业：电子与计算机工程

指导教师：朱怡飞

学院(系)：密西根学院

上海交通大学
毕业设计（论文）学术诚信声明

本人郑重声明：所呈交的毕业设计（论文），是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

作者签名：岑盛 潘飞鹏 杨迪迁 郑智睿 金哲健

日期： 2021 年 8 月 31 日

上海交通大学
毕业设计（论文）版权使用授权书

本毕业设计（论文）作者同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本毕业设计（论文）的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本毕业设计（论文）。

保密□，在___年解密后适用本授权书。

本论文属于

不保密√。

（请在以上方框内打“√”）

作者签名：岑盛 潘启颖 杨迎廷 金哲健 郑智睿　　指导教师签名：

日期：2021 年　8 月 3 日　　日期：2021 年 8 月 3 日

边缘跨摄像头视频分析

摘　要

摄像头是安保系统中十分关键的一部分。随着深度学习的发展，人们发明了智能摄像头。智能摄像头可在边缘设备（摄像头）上进行神经网络的计算，因此减少了从边缘设施到中心服务器的数据传输量。但现有研究仅将本地拍摄获得的帧作为训练数据并相关推测任务。由于不同摄像头的工作量不同，因此造成了某些空闲摄像头的算力浪费，也造成了推演任务在繁忙摄像头中的工作拥堵，极大的减小了整体视频分析系统的吞吐量。我们提出了一个新的系统可以动态调节不同摄像头的工作量。具体而言，在每个摄像头提取出每一帧图片的可用边框之后，所有的摄像头将检测见过传至服务器。服务器利用工作量平衡器将这些可用边框分配给不同的摄像头用于进一步的推演。实验显示系统在平衡工作量之后，整体的工作时间减少。

关键词：分布式深度学习系统，视频分析，工作量分配，边缘计算，跨摄像头推演

CROSS CAMERA VIDEO ANALYTICS AT EDGE

ABSTRACT

Cameras are installed widely due to surveillance consideration. With advancement of deep learning, smart cameras are introduced to train neural network models in local cameras, thus saving the data transmission cost. But previous research fed local input frames into cameras, making inference tasks independent among different cameras. It not only wastes computational resources of idle cameras, but also incurs frame congestion for busy cameras and greatly affects the system overall inference throughput. We proposed a new scheme which can dynamically balance the workloads across cameras. To be more specific, after extracting bounding boxes from video frames, all the cameras send the detection results to the server which uses a workload balancer to distribute the bounding boxes to different cameras to do further video analytics. Experiment shows the camera workload for analytics is balanced, and the overall inference runtime is shortened.

Key words: Distributed Deep Learning Systems, Video Analytics, Workload Adaptive, Scheduling, Edge Computing, Cross-camera Inference

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1 Introduction & Problem Description

1.1 Background

Video analytics has gained increasing attention in recent years. From car tracking, facial recognition to image classification, video analytics have permeated in human's life. What scientists need to improve is not only the inference accuracy of certain tasks, but also the latency and frame throughput of the processed videos.

Previously, videos were only processed in a public server (i.e. cloud computing). All the server needs to do is to gather and process the video coming from various clients (e.g. smart phones, cameras, etc.), and then outputs the inference result or gives feedback to the clients. Cloud computing has a huge power in computing due to its large capacity of CPUs, and GPUs, which does extraordinary performance in image processing. But cloud computing requires full transmission of raw video frames from the clients, consuming large bandwidth. Also, the huge transmission delay is intolerable, giving rise to edge computing.

With the advance of NPUs, more smart devices have the ability to compute locally, which is regarded as "edge computing". Edge computing makes local frame computing possible, though the computing power is limited compared with central server. Therefore, frames are not required to be transmitted to the cloud server, greatly lowering the overall delay. Also, edge computing enables multiple devices to make inference in parallel, increasing the system throughput.

1.2 Expected Outcome

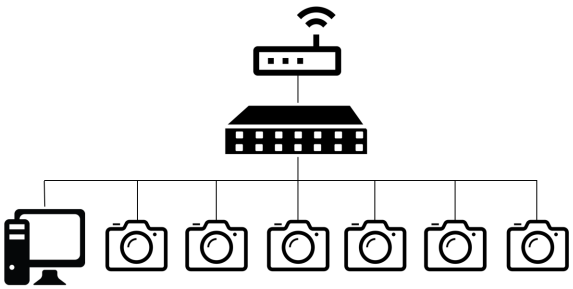


Figure 1–1 Overall Architecture of the Project

The overall architecture for our project is shown in Fig. 1–1. Multiple cameras and a server are connected through a switchboard, which is communicated through a router (shown in the top of the Fig. 1–1).

Each camera is fed into sample video frames, and is required to do inference (like image classification, facial recognition, etc.) first on the edge side (i.e. local NPU embedded in each camera). Cameras are under different workload at different times, for example, some cameras may have very few objects in a frame for certain time periods, while others may have a lot.

Therefore, our task is to balance the workload among different cameras dynamically, so that

no cameras would be in an extremely idle or busy mode on NPU running. Then, the overall frame throughput would be greatly improved. Also, we would design an algorithm which partitions workload between edges and cloud, so that the overall latency and bandwidth consumption could be balanced.

1.3 Benchmark Products

1.3.1 CloudSeg

CloudSeg framework, aiming at cloud edge in cooperation, is proposed in 2019[1]. On its edge side, the sensor (camera) adaptively down-samples a high-resolution video and streams it to the cloud server via network. On the cloud side, the server then processes the video, runs (DNN-based) inference, and finally returns the inference results to the edge device. It introduce frame skipping based on low level feature difference on the edge, and mitigating inference on the cloud.

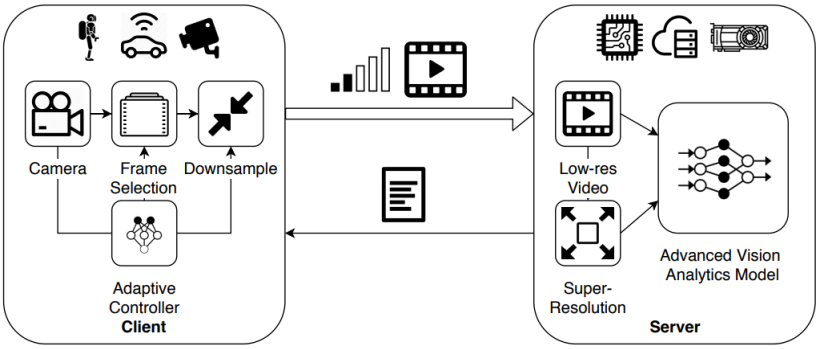


Figure 1–2 Architecture of CloudSeg

This solution can guarantee real-time advanced vision analytics over the cloud with low bandwidth consumption and speed up the processing of signals.

However, disadvantages include wrong skipping, which may lead to low inference accuracy. Therefore, if we would like to keep high accuracy, we will have to make improvements.

1.3.2 DeepDecision

In 2018, DeepDecision was proposed. It is a distributed infrastructure that ties together computationally weak front-end devices (assumed to be smartphones in this work) with more powerful back-end helpers to allow deep learning to choose local or remote execution[2].

DeepDecision makes decent collaboration of the use of server and local computing, based on a mathematical optimization problem which considers latency, inference accuracy and bandwidth usage.

But DeepDecision has no mechanism to balance the resources, or have collaboration among different cameras. Therefore, each camera is isolated from the others, and may be under workload in big rise and fall even though there's well-designed cloud-edge collaboration.

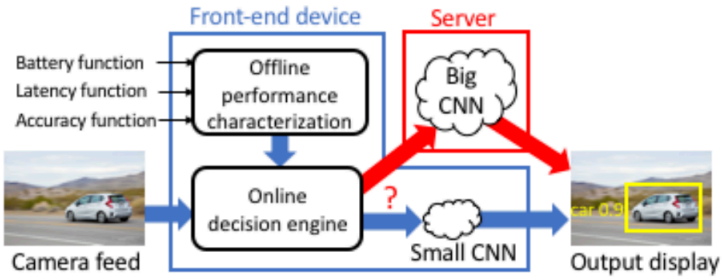


Figure 1–3 Architecture of DeepDecision[2]

1.3.3 Spatula

Spatula, proposed in 2020, focuses on a multi-camera object tracking problem[3]. It first constructs spatio-temporal correlation among different cameras offline, according to the location of different cameras. Then the query of an object is only searched from certain cameras who have a large spatio-temporal correlation with the previous frame's location of the object. Spatula greatly reduces the number of processing frames, thus lowering the latency.

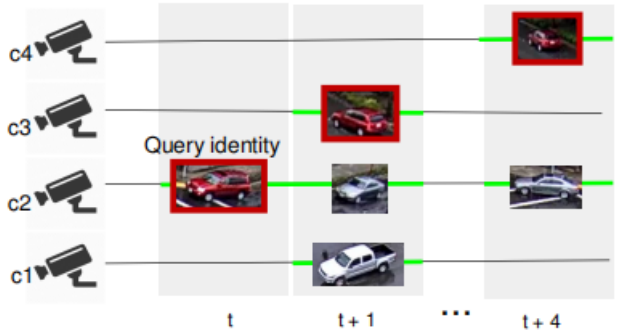


Figure 1–4 Architecture of Spatula[3]

But Spatula fully processes the frame on the edge, making the inference accuracy relatively low. Also, the spatio-temporal correlation is not fixed for different time, so wrong-skipping may happen.

1.3.4 Mainstream

In 2018, Mainstream was proposed, which makes multi-tenant joint inference using shared DNN[4]. It shares resources (partial DNN structure and parameters) among different tenants (which are cameras running different kinds of video inference tasks). The system enables multiple cameras to share the first multiple layers of computing results, avoiding repetitive but unnecessary computing steps.

But the system is only applicable for fixed sets of applications (for example, camera 1 only provides accurate result on car tracking, and camera 2 is only responsible for facial recognition). Thus. it's not adaptive to dynamic inference tasks.

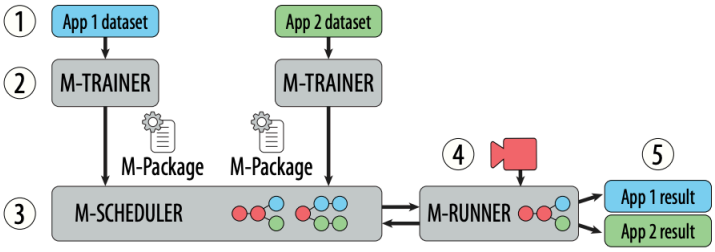


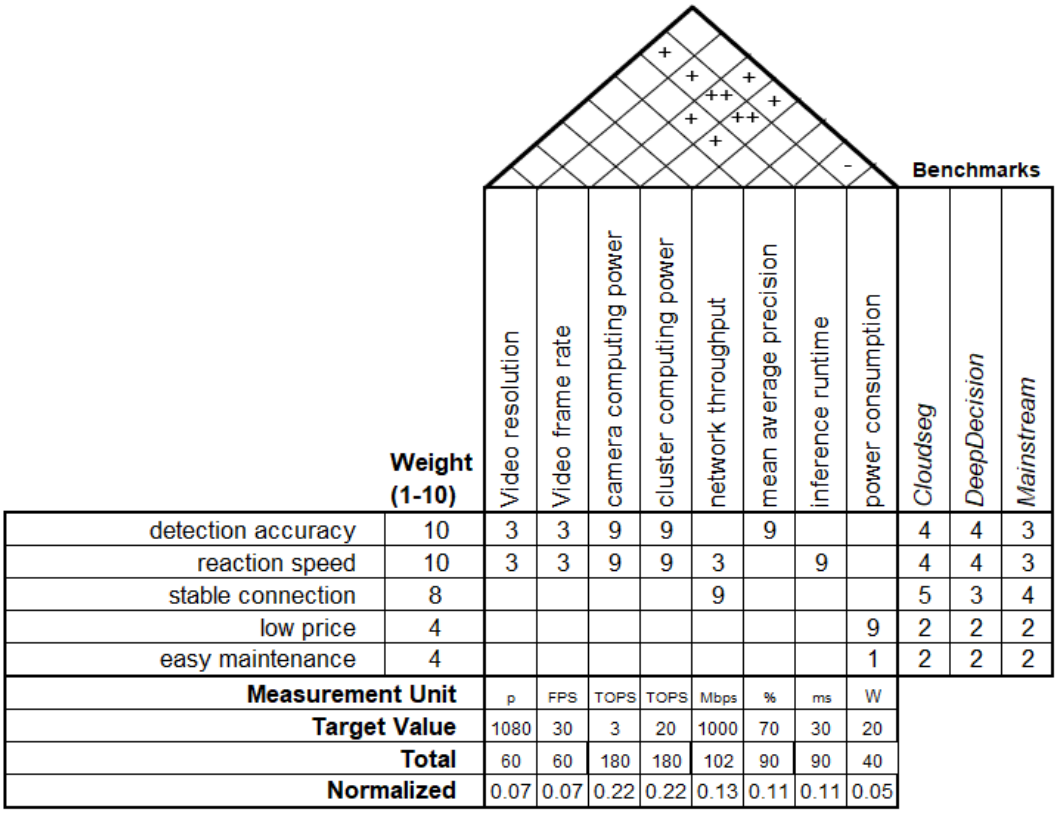
Figure 1–5 Architecture of Mainstream[4]

1.4 Novelty of Our Project

Our project aims at combining the following three specifications, dedicated to designing a cross-camera video analytics system with large frame input and low overall delay.

1. We need to design a video analytics system that fully utilizes the resources (i.e. memory, computation power) of edge and cloud.
2. We need to achieve proper workload partition between cloud and edge, achieving good trade-off between network delay and resource usage.
3. Dynamic workload allocation should be executed among cameras, improving frame throughput.

Chapter 2 Customer Requirements and Engineering Specifications



	Weight (1-10)	Video resolution	Video frame rate	camera computing power	cluster computing power	network throughput	mean average precision	inference runtime	power consumption	Benchmarks: *Cloudseg*	*DeepDecision*	*Mainstream*
detection accuracy	10	3	3	9	9		9			4	4	3
reaction speed	10	3	3	9	9	3		9		4	4	3
stable connection	8					9				5	3	4
low price	4								9	2	2	2
easy maintenance	4								1	2	2	2
Measurement Unit		p	FPS	TOPS	TOPS	Mbps	%	ms	W			
Target Value		1080	30	3	20	1000	70	30	20			
Total		60	60	180	180	102	90	90	40			
Normalized		0.07	0.07	0.22	0.22	0.13	0.11	0.11	0.05			

Figure 2–1 Quality Function Development Chart

2.1 Target market & customers

Nowadays, surveillance cameras are put into use in more and more conditions, and the scale of camera deployment can be magnificent. Besides, the application of smart cameras, which can execute part of the video analysis job, is an emerging trend to improve the surveillance effect. Traffic monitoring and campus security are two examples of application.

As an intelligent cross-camera system, we decide our target customers as the companies and government departments which run a smart surveillance camera system of wide region coverage and multiple data sources. Our design would help boost the co-operation among their cameras by maximizing the computational power with a smart conduct agent.

2.2 Customer Requirements

2.2.1 Analysis performance

For a video analysis system, what customer pays attention in the first place is the performance of the system. The performance can be evaluated from the aspects of quality and runtime, i.e., the detection accuracy and reaction speed.

The accuracy is the elementary requirement for the surveillance work. Mistakes in recognition or tracking may result in risks in the security target.

A fast reaction speed, or a low latency means the security staff on duty can better master the locations under monitoring. In traditional centralized systems, the video data would be transmitted to the cloud before being analyzed on. The large data size leads to high latency. Smart cameras in our design, move the computation to the data side and to transmit the inference result need less time due to reduced data size.

2.2.2 Stable connection

The efficient cooperation among multiple cameras requires a stable connection. To balance the workload among different devices, there will be frequent communication between them. This requests a stable network to support the and heavy information transmission load.

2.2.3 Low price

No matter for business operation or public affair, the cost of the system is an in-negligible issue. Obviously, customers would tend to choose the most cost-saving one, out of multiple solutions with similar efficiency.

2.2.4 Easy Maintenance

As a system contains large number of smart systems, the customers would require it to be user-friendly. The deployment and management of devices should be simple and efficient.

2.3 CR Weights & Benchmark Competitions

According to the importance of the CR mentioned above, we rank them as shown in Figure. 2–1. The detection accuracy and reaction speed are of the prior importance with the stable connect following. Easy maintenance and a low cost are also CR considered but are of relatively less interest in our project.

We take the cases reviewed in literature survey as benchmarks. As methods under research, most of the attention in those papers were paid to detection and network. The cloud-edge cooperation methods did well in detection jobs, with *Cloudseg* did better in connection because of the simpler structure. The scheme of *Mainstream* emphasized on the edge coordination, therefore would display a better performance on network connection. Customer requirements that were not involved are all scored 2.

2.4 Engineering specification

2.4.1 Video quality

Two major parameters used to measure the video quality is the video resolution and frame rate. The former refers to the number of pixels that a static screen is made up of, while the latter refers to the frequency of images displayed. Higher frame rate allows better dynamic details.

Both of the parameter can to some extent affect the inference accuracy, but in fact those of most modern cameras are sufficient for good inference accuracy, and therefore the relationships between these two specifications and detection accuracy are scored as medium. Besides, both of them would medially affect reaction speed.

2.4.2 Device computational power

The devices can be grouped into multiple smart cameras which working on basic video processing and an edge cluster handling the further inference as well as the workload re-distribution. The computing unit inside can be GPU/NPU, which are designed for vision tasks. Apparently the higher the computational power of those module is, the better the system performance is, i.e., better detection accuracy and reaction speed. These relationships are strong.

2.4.3 Network throughput

Throughput here defines the rate of message transmission over the local network. We hope the network can support a large enough network throughput, so that the system can work over a stable connection as well as reduce reaction time cost during data transmission.

2.4.4 Model performance

The customer would expect an excellent analysis performance, and the basic way to quantify it is to evaluate the model with some machine learning metrics. Under the context of an object detection case, mean Average Precision (mAP) is one of the common metrics.

We also hope to measure the run-time of the model. To evaluate the jobs on consecutive images series, we may calculate the inference time per query.

2.4.5 Power consumption

Except for device cost, what relates to total expense of the system is the power consumption. The power depends on devices and economical models are preferred. Light devices of lower power cost can also sightly help easy maintenance.

2.4.6 Maintenance with Docker

Easy maintenance is a requirement hard to quantify since it is a subjective feeling of using experience. Not specified in the QFD chart, it can be achieved with the adoption of docker, a platform as a service (PaaS). Instead of constructing traditional environments or virtual machines, docker delivers software in package and thus allows easy deployment/duplication and light resource consumption.

2.4.7 Supplements on ES Correlation

Engineering specifications including video quality and computational power would influence the inference runtime. Besides. the computational power affecting mAP as well as the power consumption. The runtime may as well slightly correlate with the power consumption.

2.5 Conclusion & Comments

As discussed in the part of Engineering specification, we score the CR and ES according to their correlation. The final importance rating is shown below.

Specifications	Importance		Target	
	Total	Norm.	Unit	Value
Video resolution	60	0.08	p	1080
Video frame rate	30	0.04	FPS	10
camera computing power	180	0.23	TOPS	3
cluster computing power	180	0.23	TOPS	20
network throughput	102	0.13	Mbps	1000
mean average precision	90	0.11	%	70
inference runtime	100	0.13	ms	30
power consumption	48	0.06	W	20

Table 2–1 Engineering Specifications

We see that the computational power, the network throughput and the model accuracy should be our main targets. Therefore, our project design would be focused on how to reasonably deploy the devices and models to maximize those specifications.

The targets are also given in the Table. 2–1. 1080p is the most general video resolution nowadays, while 24-30 fps are common frame rates for surveillance cameras. We hope our system would handle jobs under that circumstance. To ensure the computational power, we are selecting smart camera with NPU of about 3 tera operations per second (TOPS) and edge cluster with that of 20 TOPS. Throughput is ensured by network hardware and properly-chosen ones can ensure 1000 Mbps throughput to ensure a stable connection. We base a 70% target accuracy and 10 ms inference time per query on a review on video detection researches[5]. Besides, power consumption is estimated as 20 W for our target cameras.

Chapter 3 Concept Generation

We generate the main concept with brainstorming. To run the system, we need an appropriate video analytics pipeline, where suitable dataset and models are required. Besides, to realize the re-allocation of camera workloads, an algorithm should be designed to play the role of the workload balancer. Other aspects include the physical architecture of the system and a visualization front-end interface for system monitoring, where there is less flexibility of selection. The morphological chart Figure 3–1 offers more detailed understanding.

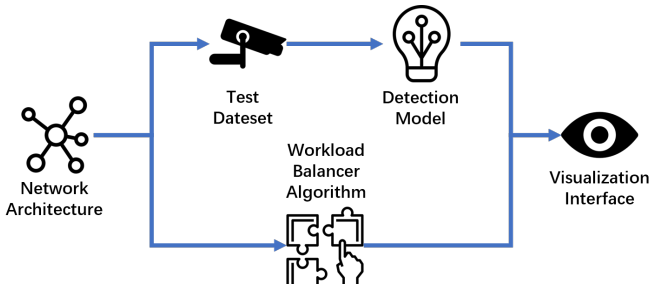


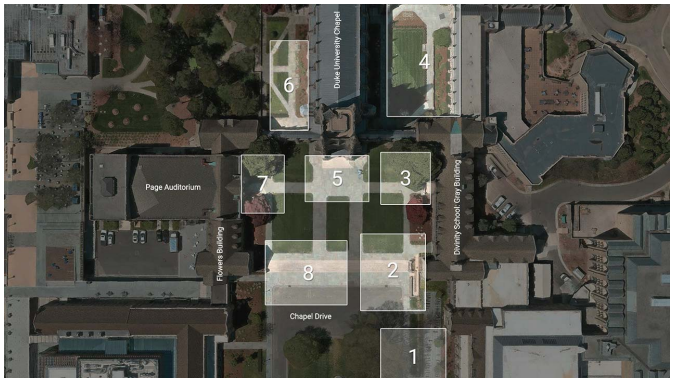
Figure 3–1 The Flowchart for Design

We are paying more attention to the selection of test dataset, detection model and the workload balancer algorithm. For the test data, we hope the video quality resembles that of monitoring cameras, and the number as well as the type of objections to be detected should be rich. There are various detection model available and we hope to find a fast, light-weighted one to fit into the edge computation scene. The workload balancer is also one of the key our project. Efficiency is of great importance due to our need for low latency.

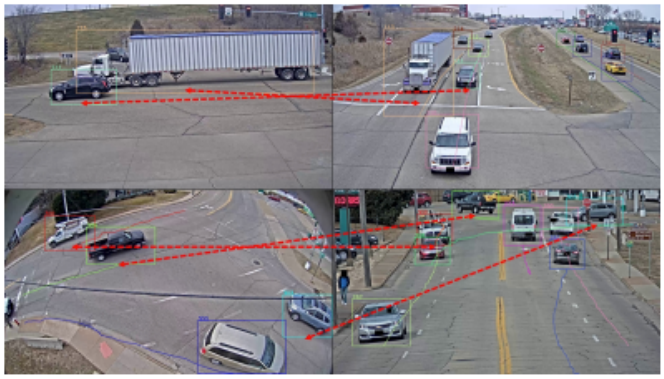
Chapter 4 Concept Selection Process

4.1 Test Dataset

To test the efficiency of our system, we need to select a proper dataset. The video dataset should be taken from multiple cameras simultaneously within a long time. A various type of objects should occur in the videos. After doing literature survey, Duke MTMC (Multi-Target, Multi-Camera)[6], CiteFlow dataset[7], VIRAT dataset[8]. The video frames are displayed in Fig 4–1a, 4–1b and 4–1c.



a) Duke MTMC Dataset



b) CityFlow Dataset

c) VIRAT Dataset

We need to select a dataset whose duration is long enough to mimic the practical situation. The video resolution and video fps should be high because of the needs of object detection and inferences. Also the dataset should include videos shot by more than five cameras so that we can test the influence of workload balancer. Based on these requirements, we make the scoring matrix in Table 4–1 and select VIRAT dataset for its highest score.

Design Criterion	Weight Factor	Unit	VIRAT			CityFlow			DUKE MTMC		
			Value	Score	Rating	Value	Score	Rating	Value	Score	Rating
Durantion	0.10	hr.	8.5	8	0.8	3.25	6	0.6	14	9	**0.9**
Scene/Camera amount	0.15	#	>10	8	1.2	40	9	**1.35**	8	7	1.05
Object Diversity	0.20	Exp	High	8	**1.6**	Low	5	1.0	low	5	1.0
Object amount	0.10	#	>30 each	8	0.8	300 ttl.	8	0.8	2000 ttl.	8	0.8
Video resolution	0.15	p	720	8	1.2	>960	9	**1.35**	1080	9	**1.35**
Video FPS	0.10	FPS	diversed	6	0.6	10	7	0.7	60	8	**0.8**
Access	0.20	Exp	public	9	**1.8**	restricted	5	1.0	denied	1	0.2
Overall Rating					8.0			6.8			5.1

Table 4–1 Scoring matrix for test dataset

4.2 Object Detection

The system needs to extract bounding boxes and determine the corresponding object types from video frames. After doing literature survey, there are three widely-used object detection models, SSD[9], Faster R-CNN[10] and YOLO[11].

According to the engineer specifications, the mAP of object detection model should be larger than 70% and the inference should be run within 100 ms. In practice, small objects occur in surveillance videos and the model need to detect them. The scoring matrix for the models is displayed in Table 4–2.

Design criterion	Weight factor	Unit	SSD			Faster R-CNN			Yolo		
			Value	Score	Rating	Value	Score	Rating	Value	Score	Rating
mean average precision	0.4	%	74.3	8	**3.2**	73.2	8	**3.2**	75	9	**3.6**
inference runtime	0.5	fpx	46	8	**4**	7	1	0.5	47	8	**4**
small object detection accuracy	0.1	EXP	Medium	6	0.6	High	9	**0.9**	High	9	**0.9**
					7.8			4.6			**8.5**

Table 4–2 Scoring matrix for object detection matrix

According to the matrix, we choose YOLO detection due to its highest score.

4.3 Workload Balance Algorithm

To handle the imbalance of the workload for heterogeneous smart cameras, an algorithm for generating workload transmission coordination is required. Firstly, an index to measure the degree of imbalance is calculated. After requesting the workload data from all the cameras in the local network, the algorithm should find a way to re-allocate the workloads, by attempting to keep the index of imbalance under some preset threshold.

Before specifying the algorithm, the index of imbalance is defined. The index is based on the device computing power as well as the instant workloads. Denoted as v, the index can be calculated as

$$v = \left(\frac{u_{max}}{\bar{u}} - 1\right)$$

where u_{max} and $\bar{u}$ respectively means the maximum and the average workloads of the camera currently. Due to the possible differences in computing power among the cameras, we also normalize the workload by multiplying a constant. That is, $u'_i = u_i a_i = u_i C_i / \bar{C}$ for the i-th camera, where C_i and C stands for the computing power of the ith camera and the average computing power. And the corresponding normalized index of imbalance is

$$v' = v = \left(\frac{u'_{max}}{\bar{u}'} - 1 \right).$$

We proposed 3 solutions for the algorithm. Firstly, the problem is obviously an optimization problem, minimizing the index of imbalance v'. Solving this problem definitely outputs the optimal distribution of workloads for the current instance. However, the main problem for it is the hardness of computation. As a nonlinear optimization problem, it is computational hard and demands higher run time as the video analytics system expands, losing the timeliness we requires. Besides, the dynamic changes in camera workload distribution means such a hard problem would be re-calculated at each time.

Therefore, a heuristic algorithm is proposed. In each iteration, simply immigrate some tasks from the camera with the highest workload to that with the lowest. Although the algorithm cannot provide with the optimal solution, it is of low computational cost and fast reaction, suitable for the edge devices with limited resources.

Finally, machine-learning-based methods proposes adopting a reinforcement learning model, to run policy optimization with gradient descend on the performance, i.e. the imbalance index. The model support long-term utility and high generality, but the conversion of the model is slow and would cost high memory.

Design Criterion	Weight Factor	Unit	Optimization			**Iterative**			RL		
			Value	Score	Rating	Value	Score	Rating	Value	Score	Rating
Reaction speed	0.5	Exp	Slow	6	2.5	Fast	9	**4.5**	Slow	7	3.5
Computational cost	0.3	Exp	High	6	1.8	Low	8	**2.4**	High	6	1.8
Solution optimality	0.1	Exp	Optimal	9	**0.9**	Non-optimal	5	0.5	Non-optimal	7	0.7
Development complexity	0.1	Exp	Medium	7	0.7	Easy	9	**0.9**	Hard	5	0.5
Overall Rating					5.9			**8.3**			6.5

Table 4–3 Scoring matrix for algorithms

We evaluate the algorithm on design criterion of reaction speed, computational cost, solution optimally as well as the development complexity. As the edge cameras are of the limited computational resources and we demands a low-latency reaction, we emphasize more the former 2 criterion, and the iterative updating algorithm thus stands out and is chosen for our design. Detailed scoring can be viewed in Table. 4–3.

Chapter 5 Final Design

5.1 Overview

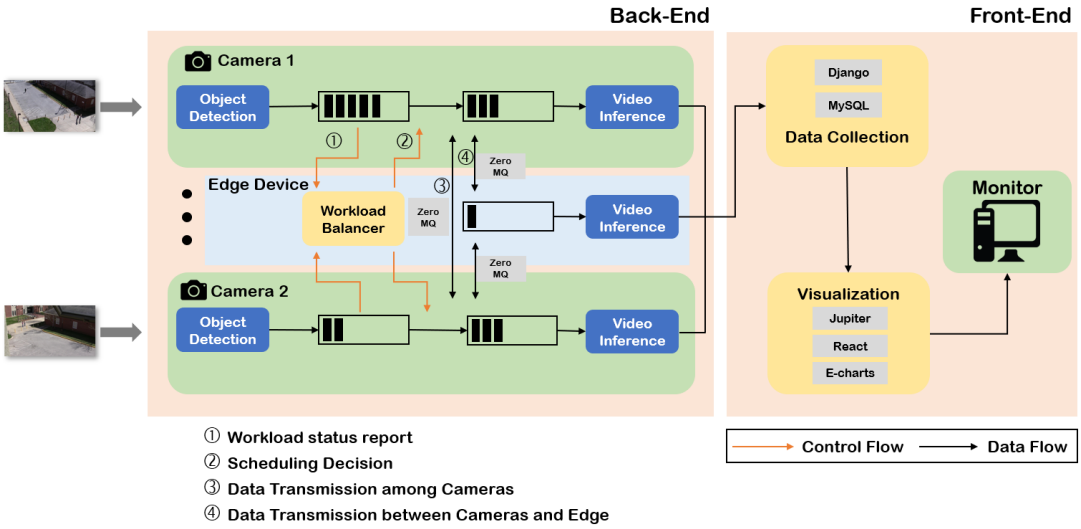


Figure 5–1 Overall Architecture of the Project

The overall system we designed consists of back-end and front-end part.

The back-end part physically consists of multiple cameras and an NVIDIA edge device. The cameras first did local object detection using YOLO. Due to the fact that we input different video clips into different cameras, each camera may have very imbalanced workload. For example, some may detect ten bounding boxes in a frame, while others may only detect one bounding box. If we did not design a cross-camera workload balancer, the overall system latency would be large, and throughput is limited by the extreme "busy" camera. Thus, we then transmit (①) the bounding box information onto the edge device, which did further workload balance and transmit (②) the instructions for balancing back to each camera. Finally, both inter-camera bounding box transmission (③) and camera/edge bounding box transmission (④) is done through ZeroMQ[12]. Each camera and edge device did inference tasks depending on the type of object in the received bounding box(es), and transmits the results to the front-end.

In this project, we accomplished cross-camera workload balance. The edge device/camera workload balance should be easy to achieve in the future research period.

The front-end part collects the inference results for each camera, and stores them in a database. Then, front-end visualization tool is utilized to depict the workload distribution among camera, CPU usage, etc.

5.2 Network Setup

Following the instruction in 6.1, we build the whole network in hte lab. The materials we used are a switch, a router, a NVIDIA Jetson Xavier NX, 6 cameras with embedded RK1808 NPU, a

tencent cloud virtual private server for intranet penetration, and several network wires.

Fig 5–2 shows the architecture of our frp model. By doing configuration in the server side and the client side, we can access the computers in LAN in our lab by SSH.

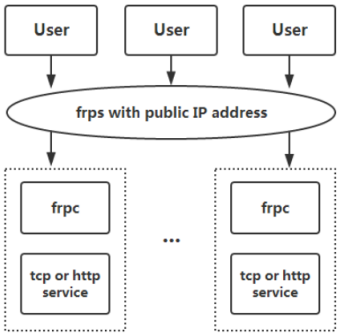


Figure 5–2 Frp model architecture

5.3 Camera video analytics pipeline

Following the instructions in 6.2, one can build the whole pipeline easily in python language. The required libraries are pytorch and tensorflow.

5.4 Camera workload balancer

The camera workload balancer takes advantage of a heuristic algorithm due to time constraint and computing ability. For each iteration it selects a pair of cameras with minimum and maximum overall workloads and determines the number of workload $x_{max,min}$ to be transferred from the maximum to the minimum based on computing capacity C_i, current workloads $w_i - \sum_j x_{ij} + \sum_j x_{ji}$ and predicted workloads $\hat{w}_i$ of all cameras. This transferring iteration continues until the imbalance condition v doesn't change anymore or it achieves a certain requirement, i.e., $v < \theta$.

Algorithm 5–1 Workload Balancer

Input: w_i: current workload of each camera; θ: balancing threshold; $\hat{w}_i$: predicted workload of each camera; C_i: computing capacity of each camera

Output: x_{ij}: number of workload to be transferred from camera i to camera j

initial $x_{i,j} = 0$ for all i, j;

$\overline{C}$ = average of C_i

$a_i = C_i/\overline{C}$ for all i **repeat**
camera i in all cameras
until for do
;

compute predicted overall workload for each camera $u_i = w_i + \hat{w}_i - \sum_j x_{ij} + \sum_j x_{ji}$;

normalize the predicted workload $u'_i = u_i a_i$;

max= arg max u; min= arg min u

compute the workload to be transferred from max to min $x_{max,min} = 0.01u'_{max}$

compute imbalance index v = max u'/ average of u'-1

$v < \theta$ or v doesn't improve anymore

Following the instructions in 6.3, one can build a workload balancer in python language with equipment of numpy library.

5.5 Quantify & Visualize

In this part we would like to visualize results and quantify the degree of optimization. We therefore need Memory (etc.) monitoring, combined with video information for intuitive visual display. Cameras will send CPU data to server every second/minute, server will store data into database, and send required data to front-end, and then display. The following picture shows a more detailed pipeline.

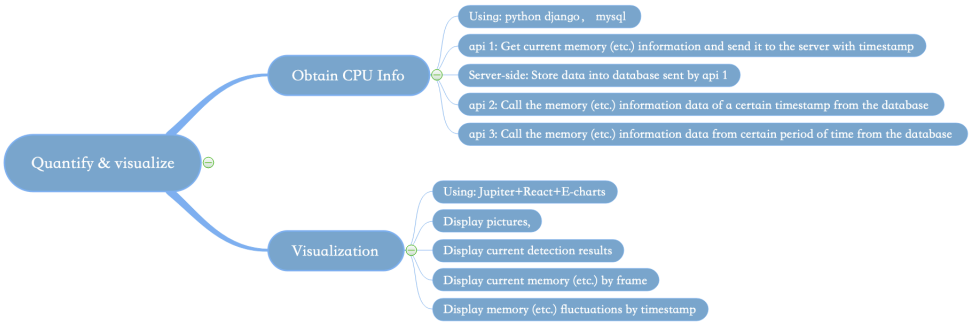


Figure 5–3 Front-end pipeline

1) Obtaining CPU info

The first API gets current memory (etc.) information and send it to the server with timestamp. Current timestamp and CPU usage is obtained by instructions, then the information needed can be

separated through regular matching and sent through API to server side. The API is triggered per minute or second. The server then stores the data into database. Another API then call the memory (etc.) information data from certain period of time from the database.

2) Obtaining Workload info

Similar to obtaining CPU info, a first API (also triggered once per second or minute) gets current workload info, then the info is stored into database, and can be called by another API.

3) Visualization

In this part we display each frame of each camera, workload before and after balancing, and CPU usage at each frame.

Chapter 6 Manufacturing/implementing Plan

6.1 Network Setup

Fig 6–1 shows the conceptual physical connection of our system. Multiple cameras are connected via a switch, which is connected to an upstream router for getting LAN. An NVIDIA edge device is also connected through the switch for the purpose of workload balance. On-site photo of part of our system is shown in Fig 6–2, where only three smart cameras are shown.

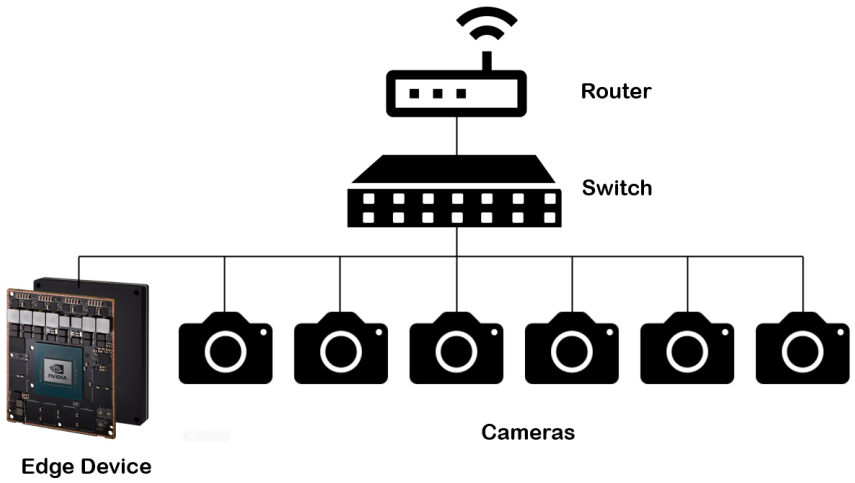


Figure 6–1 Conceptual Physical Architecture

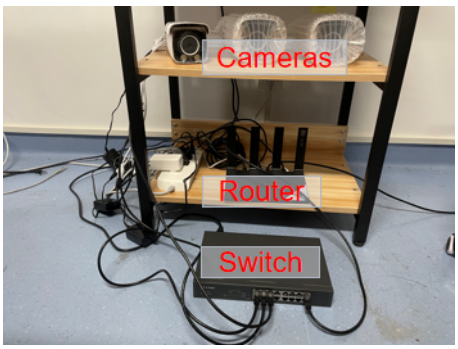


Figure 6–2 On-site Photo of Part of the System

We use basic ssh connection scheme to visit the edge-side of the cameras. Since the whole system is connected in a LAN, we could view the ip address of all the connected cameras through the administration interface of HUAWEI router website. The ip addresses for our cameras are from 192.168.3.101 to 192.168.3.106. We could visit the cameras in our lab using ssh, with username "admin" and password "202008".

When a server is not located in the lab (i.e. cannot be connected through the LAN the whole system is deployed), we could use FRP[frp-github], an open-source reverse proxy, with public ssh service

to visit our cameras remotely.

For the convenience of uploading/downloading files into/from the cameras, we download and use WinSCP, an open-source client supporting various transmission protocol, to achieve file transmission between two different devices.

We could see the interface shown in Fig 6–3 after we enter the edge-side of our smart cameras using raw ssh command. And the Winscp interface after entering the edge-side is shown in Fig 6–4.

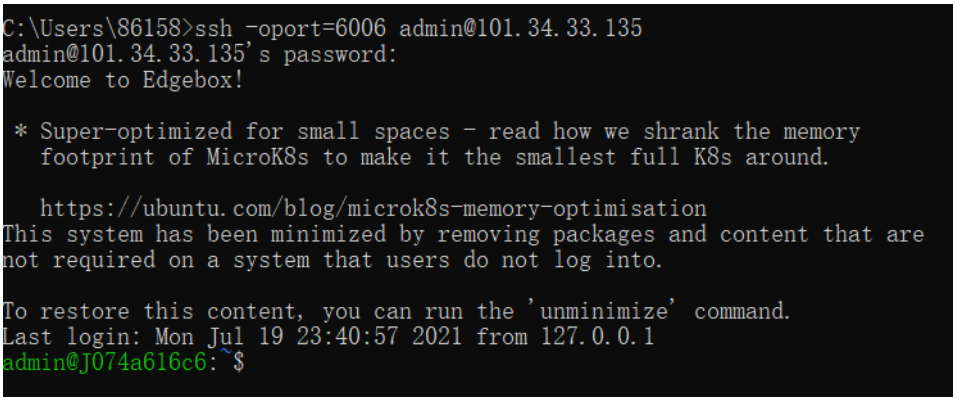


Figure 6–3 Entry Interface of Edge-sided cameras using Raw ssh Command

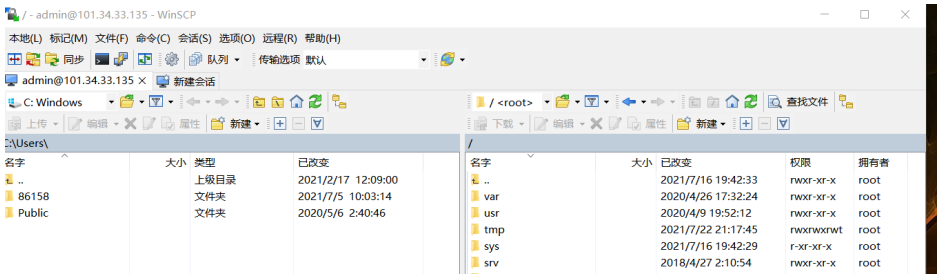


Figure 6–4 Entry Interface of Edge-sided cameras using WinSCP

6.2 Camera video analytics pipeline

For each video frame, the smart camera will first run a YOLO object detection model to extract bounding boxes from the frame. After receiving commands from the server, it will run different video inferences upon different type of assigned bounding boxes. If a bounding box display a vehicle, it will run another YOLO model to detect its type. If the vehicle type is either car or truck, a CNN model will be deployed to identify its color. Under another circumstance, a bounding box shows a person, it will just run a CNN model to recognize its cloth color. If the bounding box displays other type of objects, a new YOLO model will be used to identify whether it is an explosive. The pipeline is shown in Fig 6–5.

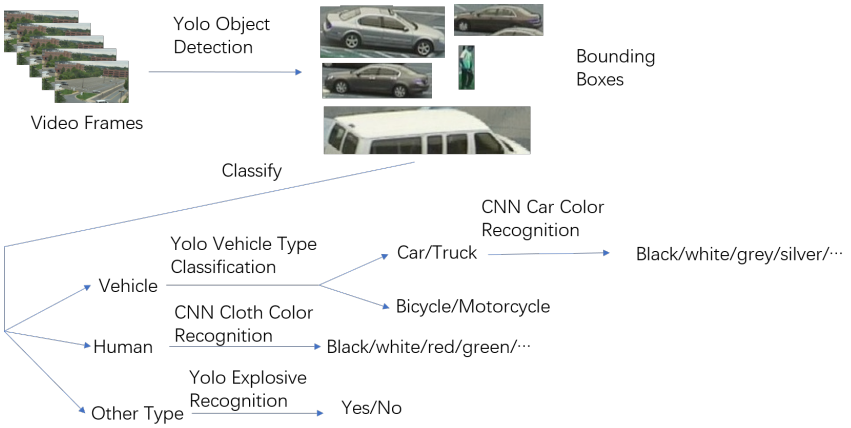


Figure 6–5 Camera video analytics pipeline

All the camera video analytics part are written in python coding language.

The YOLO model can be downloaded using commands in pytorch library as shown in Fig 6–6. The usage of YOLO model takes reference from[13].

```
import torch
yolo_model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
torch.save(yolo_model,"object_detection.pt")
```

Figure 6–6 Code for downloading YOLO model

We can modify the YOLO model to classify the vehicle type as shown in Fig 6–7

```
import torch
vehicle_model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
vehicle_model.conf = 0.1
vehicle_model.classes = VEHICLE_CLASS
torch.save(vehicle_model,"vehicle_type_classification.pt")
```

Figure 6–7 Code for downloading vehicle classification model

where the vehicle class is claimed by

```
VEHICLE_CLASS = [1, 2, 3, 7, 9]
```

Figure 6–8 Vehicle class

The numbers in the list are the corresponding indices for different type of vehicles, which are bicycle, car, motorcycle, bus and truck.

Also we can lower the confidence rate to 0.1 and save the YOLO model to identify whether an object is an explosive or not as shown in Fig 6–9.

```
import torch
explosive_model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
explosive_model.conf=0.1
torch.save(explosive_model,"others_explosive_classification.pt")
```

Figure 6–9 Code for explosive detection model

The CNN human cloth color extraction model is downloaded from[14]. This model can classify human cloth color from classes

```
classes = ['Black', 'Blue', 'Brown', 'Cyan', 'Gray', 'Green', 'More than 1 color', 'Orange', 'Pink', 'Purple', 'Red',
           'White', 'Yellow']
```

Figure 6–10 Human cloth color classes

The tensorflow model is stored in h5 format but we need to convert it into pd format to transfer into rknn model. To achieve this we can type the following code by importing keras library.

```
from keras.models import load_model
color_model = load_model('color.h5')
color_model.save('human cloth color extractor')
```

Figure 6–11 Code for transferring human cloth color extraction model

The CNN vehicle color extraction model is downloaded from[15] which implements the model[16]. The caffe model and weights are well-prepared to transfer so we don't need to change anything. It can classify color from the following ones: yellow, white, blue, cyan, red, gray, black and green.

6.3 Camera workload balancer

We realize the algorithm 5–1 in python language equipped with numpy library. According to the input of the algorithm we define the function as following

```
def distribute(workloads, threshold, predict_workload, capacity):
```

Figure 6–12 Workload Balancer Function Declaration

In the function we first initiate variables.

```
n = len(workloads)
x = np.zeros(dtype=np.int32, shape=(n, n))
c_av = np.average(capacity)
a = np.array([c / c_av for c in capacity])
v_new = None
```

Figure 6–13 Initiation in Workload Balancer Function

The variable v_new is used to check whether the imbalance index v has improved or not. After that we need to compute u, u' and v in the algorithm.

```
def calculate_u_v(x):
    u = np.array([workloads[i] + predict_workload[i] + sum(x[:, i]) - sum(x[i, :]) for i in range(n)])
    u_prime = np.array([u[i] * a[i] for i in range(n)])
    v_prime = max(u_prime) / np.average(u_prime) - 1
    return u, u_prime, v_prime
```

Figure 6–14 Computing u, u' and v

With above preparations we can code the core iteration of this algorithm and return the transferring matrix x as the output.

```
u, u_prime, v = calculate_u_v(x)
while threshold < v != v_new:
    i = u.argmax()
    j = u.argmin()
    x[i, j] += np.round(0.01 * u_prime[i])
    if v_new is not None:
        v = v_new
    u, u_prime, v_new = calculate_u_v(x)
return x
```

Figure 6–15 Code for iteration

6.4 Model deployment in Edge-side Cameras

Traditional python code of deep neural network is hard to be deployed in the edge-side cameras, due to the fact that our RK1808 camera is designed to run neural network on NPU (neural processing unit). NPU is hardware unit that is often embedded in small edge devices, like smart phones, smart cameras, etc.. NPU enjoys its reputation in fast inference time and low cost, but the inference accuracy is often low. As a result, we must first transfer our python code to the one that the camera can recognize that run on NPU.

In our project, we should first transfer our python code to the ".rknn" format which the camera could recognize. The original ".py" code is first exported to ".pt" or ".caffemodel", depending on the deep learning framework (like pytorch, or caffe,...) we used. Then the intermediate file is fed into a transferring code, which transfers the code to ".rknn" format, which is shown in Fig 6–16. The transferring code mainly invokes the interface ("rknn.api")[17], which is encapsulated in a docker image having the environment of rknn api, python 3.6, and other necessary deep learning library. Then the ".rknn" file is implanted into the cameras, to run inference task using NPU (detailed would be introduced in section 6.5.1).

```
root@32d8a8deedc2:/examples/tflite/mobilenet_v1# python ./test.py
--> config model
done
--> Loading model
done
--> Building model
W The target_platform is not set in config, using default target platform rk1808.
W:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/co
rsion.
Instructions for updating:
tf.py_func is deprecated in TF V2. Instead, there are two
    options available in V2.
    - tf.py_function takes a python function which manipulates tf eager
    tensors instead of numpy arrays. It's easy to convert a tf eager tensor to
    an ndarray (just call tensor.numpy()) but having access to eager tensors
    means `tf.py_function`s can use accelerators such as GPUs as well as
    being differentiable using a gradient tape.
    - tf.numpy_function maintains the semantics of the deprecated tf.py_func
    (it is not differentiable, and manipulates numpy arrays). It drops the
    stateful argument making all functions stateful.

W:tensorflow:From /usr/local/lib/python3.6/dist-packages/rknn/api/rknn.py:244: to
Instructions for updating:
Use `tf.cast` instead.
2021-07-21 08:40:44.842963: W tensorflow/compiler/jit/mark_for_compilation_pass.c
t.  If you want XLA:CPU, either set that envvar, or use experimental_jit_scope to
not via TF_XLA_FLAGS) or set the envvar XLA_FLAGS=--xla_hlo_profile.
W:tensorflow:From /usr/local/lib/python3.6/dist-packages/rknn/api/rknn.py:244: ad
 version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
done
--> Export RKNN model
done
```

Figure 6–16 model transfer example

The overall architecture of two-stage model transfer and inference is shown in Fig 6–17.

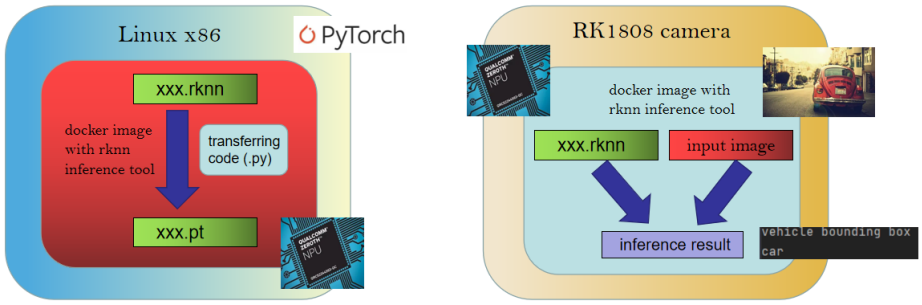


Figure 6–17 Architecture of Two-stage Model Transfer and Inference

6.5 Data Transmission

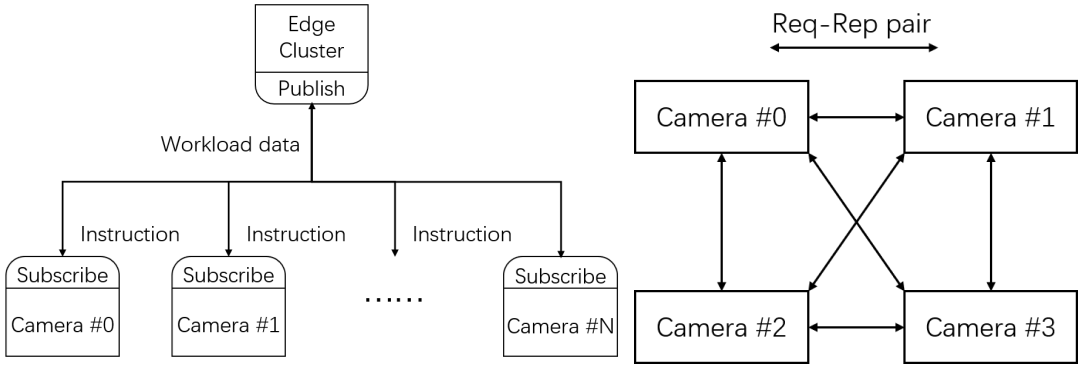
The system consists of multiple cameras as well as an edge cluster device for workload coordination, and the network scheme can thus be divided into camera-server and cross-camera connection.

For the communication between edge cluster and cameras, we design a traditional server-client paradigm, where a centralized server communicate with all the clients equally. In each iteration, the cluster runs polling requests to cameras, that is request and receive the workload data one by one. Then it would use a broadcasting scheme to send the instruction to all the camera simultaneously.

Since any one of the cameras would be sending data to another, all of the camera are of the same status, which means it should be and a peer-to-peer (p2p) network. Each camera would have a socket listening to possible incoming message, and it will connect to the target address when instructed to transmit message.

To facilitate the message queue between devices in the network, we use ZeroMQ, an open-source high-performance asynchronous messaging library, designed for distributed applications[12]. The library offers a wide range of language APIs, including Python, with which communication

programs are composed. ZeroMQ-contexted network schematic diagrams are shown in Figure 6–18a and 6–18b.



a) Cluster-Camera network

b) Cross-Camera network

Figure 6–18 Network schematic diagrams

At initialization, the server program create a publisher socket for instruction publishing, then creates requester sockets, one for each camera and connects to it. In the main loop part, the server requests the workload data and store locally. Then the balancing algorithm produce instruction, which are later broadcast in the network.

```
context = zmq.Context()
# create PUBLISHER socket, binding to port 50000
instr_publisher = context.socket(zmq.PUB)
instr_publisher.bind("tcp://*:50000")
# create REQUESTER sockets, connecting to ports 5000X
workload_queriers = [context.socket(zmq.REQ)] * CAMERA_NUM
for i, skt in enumerate(workload_queriers):
    skt.connect("tcp://192.168.3.{}:{}".format(101+i, 50001+i))

workloads = [0] * CAMERA_NUM
```

a) Server Initialization

```
while 1:
    # request workload from each camera
    for i, skt in enumerate(workload_queriers):
        skt.send_string('Hi')
        message = skt.recv_json()
        workloads[message['id']] = message['data']

    instr = balancer(workloads)

    # Synchronize, and send instruction
    time.sleep(5)
    instr_publisher.send_json(instr)
```

b) Server Main Loop

Figure 6–19 Server Code

On the other hand, the client program creates socket for sending or receiving messages. There are two threads running at the same time, one for communication with the cluster and the other for that with cameras. In workload reporter loop, it reports the workload data on receiving request from the cluster. In data receiver loop, it simply waiting for incoming message from other cameras.

```python
# create RESPONDER socket, binding to port 5000X
# reporting workload to server
workload_reporter = context.socket(zmq.REP)
workload_reporter.bind("tcp://*:{}".format(50001+CAMERA_ID))

# create SUBSCRIBER socket, connecting to port 50000
# receiving instruction from server
instr_receiver = context.socket(zmq.SUB)
instr_receiver.connect("tcp://192.168.3.201:50000")
instr_receiver.subscribe("")

# create RESPONDER socket, binding to port 40001+camera_id
# listening message from other cameras
recv_socket = context.socket(zmq.REP)
recv_socket.bind(f"tcp://*:{40001 + CAMERA_ID}")
# create REQUESTER socket
# sending message to other camera
send_socket = context.socket(zmq.REQ)
```

a) Client Initialization

```python
while 1:
    assert workload_reporter.recv_string() == 'Hi'
    message = {'id': CAMERA_ID, 'data': get_workload()}
    workload_reporter.send_json(message)
    # print("send {}".format(message['data']))

    instr = instr_receiver.recv_json()
    # print(instr)
    send_skt(send_socket, instr)
```

b) Client Workload Reporter Loop

```python
while 1:
    message = socket.recv_string()
    print(message)
    socket.send_string("0")
```

c) Client Data Receiver Loop

Figure 6–20 Client Code

6.5.1 NPU Model Inference with Workload Balance

We designed a two-step inference architecture (Fig 6–21), deployed in the camera.

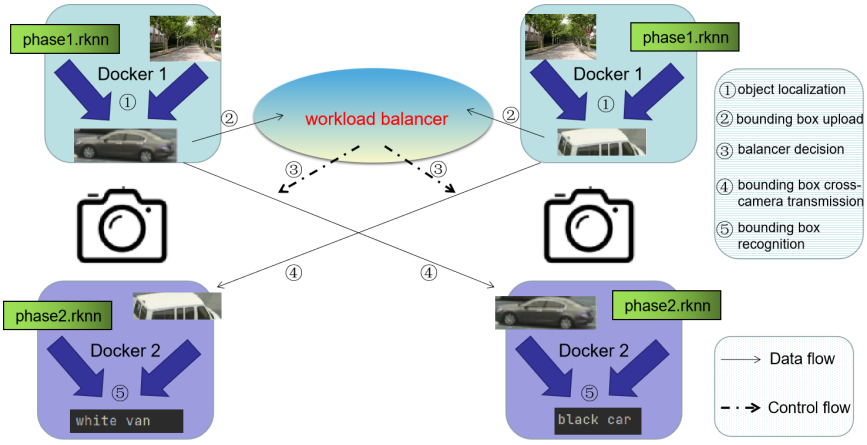


Figure 6–21 Cross-camera Two-step Inference Architecture with Docker

Each camera is injected into two dockers, with an rknn-toolkit image. The first docker generates (①) bounding box(es) in a local video frame and reports the type of the objects. These information is then transmitted (②) to the workload balancer in the central edge device. The workload balancer then run the algorithm to generate (③)an instruction of allocating the bounding boxes to each camera. Then the bounding boxes are transmitted (④) among cameras. Finally, each camera uses the second

docker to do (⑤) analytics according to the new bounding box(es) it receives and the reported types of object belonging to each bounding box.

6.6 Quantify & Visualize

In this part we visualize results and quantify the degree of optimization. The pipeline here can be separated into 3 parts, obtaining CPU info, obtaining workload info, and visualization. All API for obtaining CPU info are written in Python using Django.

1) Obtaining CPU info

For the first API, current timestamp and CPU usage are obtained through instructions, then the information needed can be separated through regular matching and sent through API to server side. The code of the first API is shown below. Sample output includes time, total CPU, free CPU, used CPU and buffer. If error occurs, it returns error.

```
def getCpu(request):
    if request.method == 'POST':
        receive_data = json.loads(request.body.decode())
        mode = receive_data['mode']
        if not mode:
            resp = {'errorcode': 100, 'tos': '', 'detail': 'Need Mode'}
            return HttpResponse(json.dumps(resp), content_type="application/json")
        if mode == 'CPU':
            f = os.popen('top -b | head -50')
            str = f.read()
            cpu = re.search(r'Mem : (.*) total, (.*) free, (.*) used, (.*) buff/cache', str, re.M
            total = 0
            free = 0
            used = 0
            buff = 0
            if cpu:
                # print (cpu.group(0))
                total = cpu.group(1)
                free = cpu.group(2)
                used = cpu.group(3)
                buff = cpu.group(4)
            resp = {'errorcode':0, 'time':time.time(),'total':total,'free':free,'used':used,'buff'
            return HttpResponse(json.dumps(resp), content_type="application/json")
    return HttpResponse("This is GET. Use POST")
```

Figure 6–22 Codes for Obtaining CPU info

The server side will store every message sent by the first API if no error occurs.

```
function GetCpu(req) {
  const uri = `http://#ip-address:8000/image/get_cpu`;
  const body = JSON.stringify(req);

  return fetch(uri, { method: 'POST', body });
}

async function getCpu() {
  try {
    const res = await GetCpu({
      mode: 'CPU',
    });
    const data = await res.json();
    const newitem = cpu_table.create({
      t: data.time,
      time: data.time.toFixed(),
      cpu_used: Number(data.used),
      cpu_total: Number(data.total),
      cpu_free: Number(data.free),
      cpu_buff: Number(data.buff),
    });
    await cpu_table.save(newitem);
    return data;
  } catch (e) {
    return e;
    // console.log(e);
  }
}
```

Figure 6–23 Codes for storing info

The last API filters data in database and returns data in a certain period. Input here is two timestamps and sample output is a map with each node including information stored by API 1.

```
for (let i = Number(start); i <= Number(end); i += 1) {
  const cpu = await cpu_table.where({ time: i.toString() }).findOne();
  // console.log(i);
  // console.log(cpu);
  if (cpu) {
    res.push({
      cpu_used: cpu.cpu_used,
      cpu_total: cpu.cpu_total,
      cpu_free: cpu.cpu_free,
      cpu_buff: cpu.cpu_buff,
      time: new Date(Number(cpu.time) * 1000 + 8 * 60 * 60 * 1000),
    });
  }
}
```

Figure 6–24 Codes for fetching data

2) Obtaining Workload info

Similar to obtaining CPU info, a first API (also triggered once per second or minute) gets current workload info, then the info is stored into database, and can be called by another API. The codes here are also similar.

3) Visualization

In this part we display each frame of each camera, workload before and after balancing, and CPU usage at each frame. We use React Hooks for general front-end developing, and React E-charts for graphs.

Codes for displaying each frame of each camera. Note that pagination here is the control for choosing frames.

```jsx
<h1 style={{ position: 'relative', top: 20, left: 20 }}>Camera 1</h1>
<Pagination
  total={200}
  style={{ padding: 20 }}
  onChange={v => {
    setSelectFrame(v);
  }}
/>
<img
  style={{ padding: 20, width: '90%' }}
  src={imgData[selectFrame - 1]}
  alt=""
/>
<h2 style={{ padding: 20 }}>
  Result will be later labeled on the graph{' '}
</h2>
```

Figure 6–25 Codes for choosing frames

Codes for getting CPU data from API (Similar for getting workload data):

```jsx
function GetCpu(req) {
  const uri = `https://#ip_address/get_cpu_period`;
  const headers = { 'Content-Type': 'application/json' };
  const body = JSON.stringify(req);
  // const body = toParams(req)
  return fetch(uri, { method: 'POST', headers, body });
}
async function tryapi() {
  try {
    const res = await GetCpu({
      time_a: '#sample time 1',
      time_b: '#sample time 2',
    });
    const data = await res.json();
    const tmpTime = Array(0);
    const tmpUsed = Array(0);
    const tmpFree = Array(0);
    const tmpBuff = Array(0);
    const tmpTotal = Array(0);
    // console.log(data);
    data.cpu_period.map(item => {
      tmpTime.push(item.time.split('.')[0]);
      tmpUsed.push(item.cpu_used);
      tmpFree.push(item.cpu_free);
      tmpBuff.push(item.cpu_buff);
      tmpTotal.push(item.cpu_total);
    });
    setTime(tmpTime);
    setUsed(tmpUsed);
    setFree(tmpFree);
    setBuff(tmpBuff);
    setTotal(tmpTotal);
    // console.log(tmpTime);
    return data;
  } catch (err) {
    return err;
  }
}
```

Figure 6–26 Codes for getting data from API

Codes for showing workload (Similar for showing CPU data):

```
<ReactEcharts
  style={{ padding: 20, height: 400 }}
  option={{
    title: {
      text: 'Workload at each frame',
    },
    legend: {},
    tooltip: {},
    dataset: {
      dimensions: ['product', 'Before Balance', 'Balanced'],
    },
    xAxis: {
      type: 'category',
      data: ['Camera 1', 'Camera 2', 'Camera 3', 'Camera 4'],
    },
    yAxis: {},
    series: [
      {
        name: 'Before Balance',
        data: dataBeforeBalance,
        type: 'bar',
      },
      {
        name: 'Balanced',
        data: dataAfterBalance,
        type: 'bar',
      },
    ],
  }}
/>
```

Figure 6–27 Codes for showing workload

Chapter 7 Validation Results

7.1 Camera video analytics pipeline

It is a waste of time to manually determine the inference results of all the video frames from VIRAT dataset. Ten video frames are selected randomly and we manually detect objects and classify their colors and vehicle type if vehicles exist. Then we run the inferences on a smart camera. We record the inference runtime and calculate the mean average accuracy. The detailed results are displayed in Fig 7–1.

	Yolo Object Detection	Yolo Vehicle Type Classification	CNN Car Color Recognition	CNN Cloth Color Recognition	Yolo Explosive Recognition
Mean Average Precision(%)	75	90	82	83	100
Inference runtime(ms)	44	35	13	14	29

Figure 7–1 Inference result

7.2 Camera Workload Balancer

To test the efficiency of this balancer, we can generate random 20-round workloads for 6 cameras with different mean. Comparing the workload distribution with and without balancer, we can conclude that the heuristic algorithm satisfies the engineer specification.

Round	1	2	3	4	5	6	7	8	9	10
Var of workloads without balancer	27.25	46.14	268.89	514.58	978.47	1990.67	2675.14	3981.56	4201.67	5553.89
Var of workloads with balancer	16.25	29.56	90	196.47	387.33	753.89	1581.58	2064.22	2862.14	2927.92
Workload Var Decreasing Rate	0.40367	0.359341	0.665291	0.618193	0.604147	0.621288	0.408786	0.481555	0.318809	0.472816
Round	11	12	13	14	15	16	17	18	19	20
Var of workloads without balancer	7479.67	9701.14	10888.33	11227.81	11765.25	14041.81	16189.81	17904.56	17236.22	21498.89
Var of workloads with balancer	3826.47	5206.92	6381.89	7181.58	6932.56	6941.33	8559.22	9580.89	10502.81	9451.14
Workload Var Decreasing Rate	0.488417	0.463267	0.413878	0.360376	0.41076	0.505667	0.471321	0.464891	0.390655	0.560389

Figure 7–2 Test results for workload balancer

A plot can vividly display the effects of camera workload balancer.

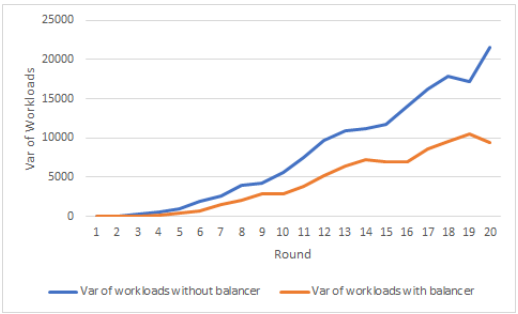


Figure 7–3 Plot for data in Fig. 7–2

In practice, the camera workload balancer also greatly reduces the variance of the system.

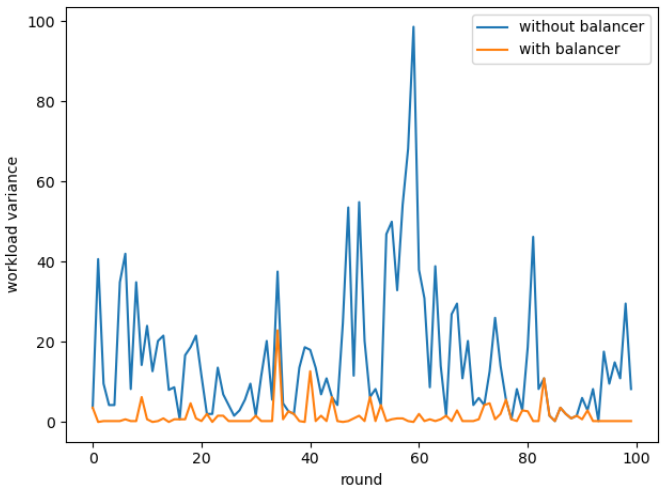


Figure 7–4 Plot for data in Fig. 7–2

7.3 The overall system performance

We use 3 cameras to build this system. The time for 3 cameras to accomplish their corresponding workload pipeline each round is shorted when workload balancer is deployed.

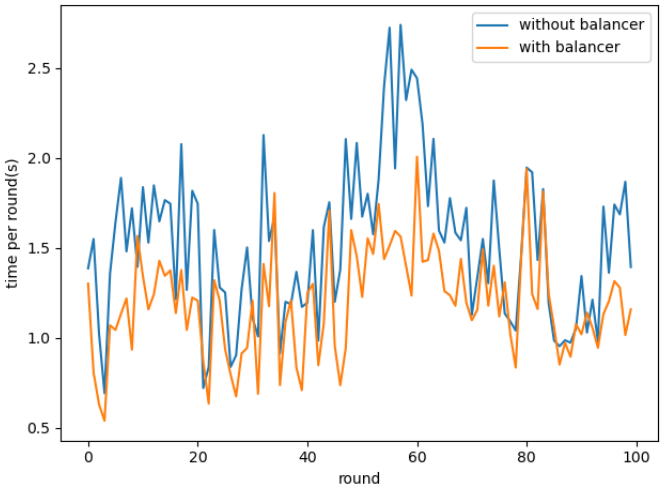


Figure 7–5 Plot for runtime per round

The accumulated time plot is shown as follows. Workload balancer declines the overall runtime.

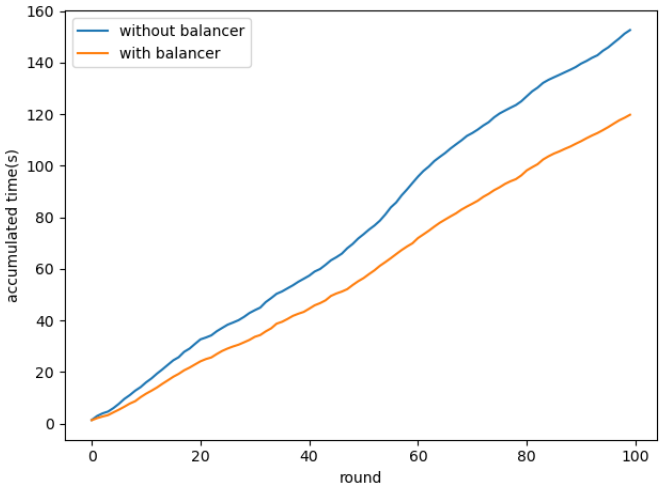


Figure 7–6 Plot for overall runtime

The throughput measured by the number of inference time per second also shows the improvement of our system.

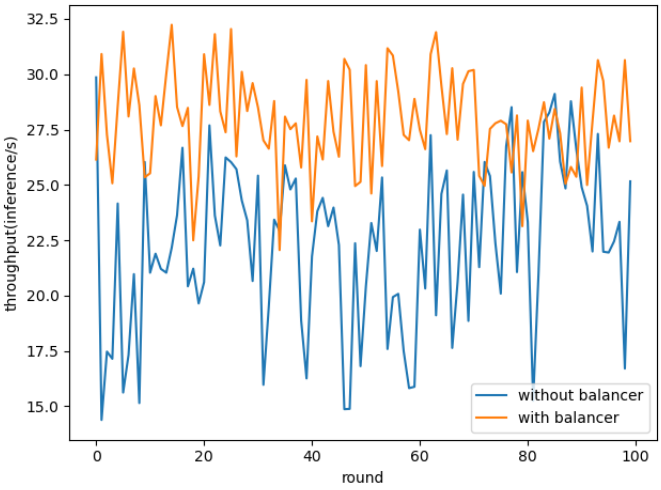
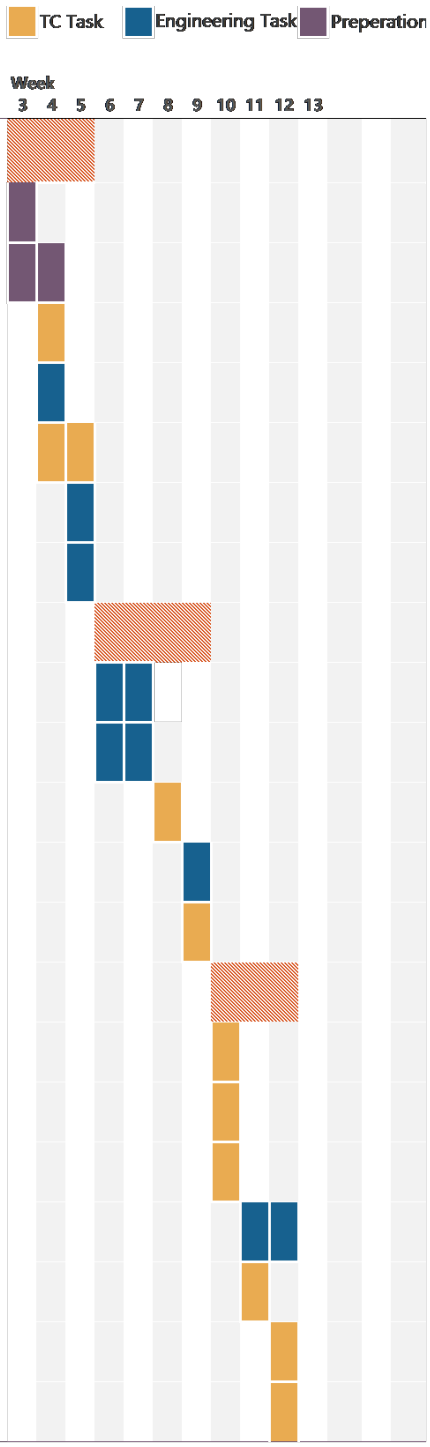


Figure 7–7 Plot for throughput

Chapter 8 Project Timeline and Plan



Task	Starting time	Period	Team members involved
Milestone 1: Preperation	**3**	**2**	**All**
Design problem and needs determination	3	1	All
Literature review	3	2	All
Design review#1 pre	4	1	All
Cross-camera communication	4	1	Cen & Jin
Design review #1 report	4	2	Pan, Yang & Zheng
Dataset collection	5	1	Pan
Local video analysis	5	1	Pan & Yang
Milestone 2: Algorithm Design	**6**	**4**	**All**
Cross-camera workload balancer	6	2	Cen & Zheng
Camera-cluster workload partitioner	6	2	Pan & Jin
Design review #2 pre	8	1	All
Front-end design	9	1	Zheng
Design review #3 pre	9	1	All
Milestone 3: Design Integration	**10**	**3**	**All**
Design review #3 report	10	1	Cen & Yang
Design expo poster	10	1	Pan
Design expo video	10	1	Jin & Zheng
Design integration and improvement	11	2	All
Final thesis version 1	11	1	Cen & Yang
Final pre	12	1	All
Final thesis	12	1	All

Figure 8–1 Project plan

Chapter 9 Discussion

Our system is an intelligent edge video analytic system, which adopt dynamic allocation of camera workload to boost the overall system performance. We combine the traditional local network of monitoring cameras with the novel ideas of edge computing. A specialized system architecture and a practical algorithm are implemented to realize the scheme. Tests show the system achieves high throughput and low latency on a local camera network, reaching a satisfying tradeoff between resource usage and the network latency, representing a new solution for systematic video analytics solution.

Our design still has some points that are not ideal. Firstly, to simplify the testing steps, we choose to run inference on video datasets, which makes the cameras themselves got reduced to just computers. The involvement of camera stream for instant video analysis can make the simulation more reliable. Besides, because we are new to the NPU developments, the capacity of the hardware may not be fully deploited. More solid knowledge of computer hardware can help improve the overall performance and efficiecy.

Chapter 10 Conclusions

In conclusion, we have designed a video analytics system that dynamically distributes workloads across smart cameras with the assist of an edge cluster. We aim to lower the latency and avoid queue congestion by maximizing the utility of all the computing resources in the system compared to the benchmarks. Our validation result shows that we have increased network throughput, balanced the cross-camera workload, and guaranteed the inference accuracy. Detailed schedule is displayed in 8–1 so that we can follow the plan and complete the project.

Chapter 11 Future Works

One potential problem is that the algorithm model may not be adaptive to our NPU because our smart camera uses aarch64 architecture. To solve this problem, we need to choose the platform dependencies carefully.

Another potential problem is that the prototype that we realized has to do two-stage deep learning inference subsequently, i.e. our prototype need to do the object detection first, then it can do the video inference after that. However, if we can do the two module together, we can decrease the total inference time. To solve this problem, we need to investigate whether the NPU can do the two stages concurrently.

Bibliography

[1] DEV S, NAUTIYAL A, LEE Y H, et al. Cloudsegnet: A deep network for nychthemeron cloud image segmentation[J]. IEEE Geoscience and Remote Sensing Letters, 2019, 16(12): 1814-1818.

[2] RAN X, CHEN H, ZHU X, et al. Deepdecision: A mobile deep learning framework for edge video analytics[C]//IEEE INFOCOM 2018-IEEE Conference on Computer Communications. [S.l. : s.n.], 2018: 1421-1429.

[3] JAIN S, ZHANG X, ZHOU Y, et al. Spatula: Efficient cross-camera video analytics on large camera networks[C]//2020 IEEE/ACM Symposium on Edge Computing (SEC). [S.l. : s.n.], 2020: 110-124.

[4] JIANG A H, WONG D L K, CANEL C, et al. Mainstream: Dynamic stem-sharing for multi-tenant video processing[C]//2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18). [S.l. : s.n.], 2018: 29-42.

[5] ZHU H, WEI H, LI B, et al. A Review of Video Object Detection: Datasets, Metrics and Methods[J]. Applied Sciences, 2020, 10(21): 7834.

[6] RISTANI E, SOLERA F, ZOU R S, et al. Performance Measures and a Data Set for Multi-target, Multi-camera Tracking[C]//ECCV Workshops. [S.l. : s.n.], 2016.

[7] TANG Z, NAPHADE M, LIU M Y, et al. Cityflow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. [S.l. : s.n.], 2019: 8797-8806.

[8] OH S, HOOGS A, PERERA A, et al. A large-scale benchmark dataset for event recognition in surveillance video[C]//CVPR 2011. [S.l. : s.n.], 2011: 3153-3160.

[9] LIU W, ANGUELOV D, ERHAN D, et al. SSD: Single Shot MultiBox Detector[J]. Springer, Cham, 2016.

[10] REN S, HE K, GIRSHICK R, et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2017, 39(6): 1137-1149.

[11] REDMON J, DIVVALA S, GIRSHICK R, et al. You Only Look Once: Unified, Real-Time Object Detection[J]. IEEE, 2016.

[12] https://zeromq.org/.

[13] https://pytorch.org/hub/ultralytics_yolov5/.

[14] ZeynepCankara. ZeynepCankara/Clothing-Style-Detector: Clothes style detector, predicts patterns and color together with the clothing category.[EB/OL]. https://github.com/zeynepCankara/Clothing-Style-Detector.

[15] Rezafuad. Rezafuad/vehicle-color-recognition[EB/OL]. https://github.com/rezafuad/vehicle-color-recognition.

[16] RACHMADI R F, PURNAMA I. Vehicle color recognition using convolutional neural network[J]. ArXiv preprint arXiv:1510.07391, 2015.

[17] Radxa. https://github.com/rockchip-linux/rknn-toolkit/.

Appendix A Bios

- Sheng Cen (Team Captain)

I am a senior student majoring in ECE in JI. I'm interested in the area of computer network and computer science. Four years' undergraduate period helps me lay a solid foundation in advanced mathematics, principles in programming, database system, fundamental machine learning algorithms, data structures, idea of reinforcement learning, communication theory, etc.. I would pursue my PhD degree in JI, under the supervision and guidance of Prof. Yifei Zhu. My future research orientation would be edge computing, video analytics system, and federated learning. In the future, I would try my best to become an ambitious researcher, or a respectable professor in college.

In this project, I would be in charge of the overall architecture of the system, and trying to embed our novel scheduling algorithm in the project. Also I would be responsible for reimbursement, major communication among instructor and teammates, as well as the organization of each activity in this course.

- Qiying Pan

I am a senior student majoring in ECE. I will take part of the algorithm design in this project. I divide the project into three optimization problems. All the three need well-defined mathematical models. For smart camera workload balancer, over-balancing problem should be considered to lower the costs of transferring data across camera. For camera-edge cluster partitioner, the decision variables should be determined in a finite set as we cannot distribute

one work to different devices. For workload adaptation controller we need to design the objective function in system-level as only the overall performance of the edge cluster and the cameras is considered. I plan to finish these three algorithm designs one by one. After that, I will cooperate with other teammates to realize them on hardwares.

- Zhejian Jin

I am a senior student at UM-SJTU Joint Institute majoring in ECE. I have learnt programming languages including C, C++, Python, Golang, etc. I'm familiar with operating systems, deep learning and reinforcement learning. I'm going to further complete my master degree at Columbia University.

In this project, I'll do the back-end development and system architecture design part. With my teammates, we'll set up the local area network among the PC and the cameras. I'll also do optimizations on the network and workload in order to get a better overall performance.

- Xuting Yang

I am a senior ECE student in UM-SJTU Joint Institute. The education at JI offers me plenty of development chances with programming languages including C/C++, Python, R as well as web design. I am also familiar with different fields like computer architecture, algorithm and data structure, machine earning and AI, statics etc. I am now preparing to study further in computer engineering at University of Southern University in the coming fall.

In this project, I would be responsible for the back-end development. Together with my reliable teammates, I will set up the devices and deploy necessary software and programs, with which the communication and cooperation between multiple devices can be realized.

- ZhiRui Zheng

I am a senior ECE student at UMJI-SJTU. Programming languages I master include C/C++, JavaScript, Python, Golang, R, etc. I will take part in the algorithm part of designing the workload balancer and I will also be responsible for front-end design. For the front-end part I decide to use Jupiter as framework and use react as library for building user interfaces. The website will show the original video as well as our processed video, and it will also show information such as current delay and memory usage.

Appendix B Comments from Judge Panels

B.1 Comments from Design Review 1

Question 1: Will your group deploy real cameras? In other words, is your group a pure software group?

Answer 1: We have purchased six real smart cameras. Dataset would be loaded into the edge side of the cameras, and related experiments would be carried out. Detailed experimental setup is shown in 6–2.

Question 2: In slide 9, what does ”easy maintenance” mean?

Answer 2: We would use docker to encapsulate the running environment of a certain code. The environment could be implanted into different operating systems. Therefore, we could develop and run these codes easily in different running environment, which is easier for developer to maintain the codes. The concrete benefit could be shown in section 6.4.

B.2 Comments from Design Review 2

Question 1: In slide 20, how does the detailed ”Score” calculated from the categorized ”Value”?

Answer 1: The ”Value” is categorized into different levels, and we can get the ”Score” in rough transfer. More information can be found in section 4.3

Question 2: In slide 12, why do you want to choose only one dataset?

Answer 2: For the dataset we have chosen, it has already contained many cameras' view. And the workload for each camera is not balanced, so we could utilize it, which is shown in 4.1.

B.3 Comments from Design Review 3

Comment: In the slide 13, the detailed algorithm should not be listed in the slide. Audience of a presentation could not catch the general idea from a detailed algorithm. Instead, some more vivid illustrations should be given.

Question 1: In the demonstration video, why some cameras are covered with plastic covering? Will they take real-time pictures in EXPO?

Answer 1: For ease of system testing purpose, we inject video dataset into each camera to simulate real video footage at current stage. So no need to actually use these cameras to take videos. But as soon as we finished the whole system building process, it will be easy to deploy them in the real world. Actually these cameras are already working, but since our system is a networked camera system, for demo purpose, in the capstone, we will not really install them on the wall or at different places, we will just put them together. Actually, the dataset is taken from real life, and the information can be found in section 4.1.

Question 2: Is your camera virtual camera?

Answer 2: Definitely they are not virtual camera. They are real cameras and smart cameras that have computing power inside.

Appendix C ECN

In design review 3, we plan to manufacture a parallel system that object detection and further inference are run in parallel. But we find that this needs much bigger cumputing resources than expected and our smart cameras cannot meet this requirement. So we change the parallel system to sequential system. That is, after the edge cameras extract bounding boxes from their local frames, they wait for the instructions from the server to do further inferences. When all the cameras finish their inferences, they will then detect objects from other video frames.

Appendix D Bill of Materials

Quantity	Part Description	Purchased From	Part Number	Price (yuan each)
1	UGREEN CAT6 Gb connection wire	Shanghai Yuanmai Trading Co.,Ltd	20159	7.88
2	TPLINK16 Gb exchanger connection wire	Shanghai Yuanmai Trading Co.,Ltd	TL-SG1016DT	353.10
1	HUAWEI router 4G	Shanghai Yuanmai Trading Co.,Ltd	B311B-863	308.85

Figure D–1 Bill of Materials

Appendix E Related Codes

E.1 Camera workload pipeline

```
import torch
yolo_model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
torch.save(explosive_model,"object_detection.pt")


from keras.models import load_model
color_model = load_model('color.h5')
color_model.save('human cloth color extractor')


import torch
explosive_model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
explosive_model.conf=0.1
torch.save(explosive_model,"others_explosive_classification.pt")



import torch
VEHICLE_CLASS = [1, 2, 3, 7, 9]
vehicle_model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
vehicle_model.classes=VEHICLE_CLASS
vehicle_model.conf=0.1
torch.save(vehicle_model,"vehicle_type_classification.pt")
```

E.2 Camera workload balancer

```
import numpy as np

def distribute(workloads, threshold, predict_workload, capacity):
    def calculate_u_v(x):
        u = np.array([workloads[i] + predict_workload[i] + sum(x[:, i])
        - sum(x[i, :]) for i in range(n)])
        u_prime = np.array([u[i] * a[i] for i in range(n)])
        v_prime = max(u_prime) / np.average(u_prime) - 1
        return u, u_prime, v_prime

    n = len(workloads)
```

```
    x = np.zeros(dtype=np.int32, shape=(n, n))
    c_av = np.average(capacity)
    a = np.array([c / c_av for c in capacity])
    v_new = None
    u, u_prime, v = calculate_u_v(x)
    while threshold < v != v_new:
        i = u.argmax()
        j = u.argmin()
        x[i, j] += np.round(0.01 * u_prime[i])
        if v_new is not None:
            v = v_new
        u, u_prime, v_new = calculate_u_v(x)
    return x
```

E.3 Main part for server

```
import zmq, time
from . import util

CAMERA_NUM = 5  # number of camera in the network

def main():
    context = zmq.Context()
    instr_publisher = context.socket(zmq.PUB)
    instr_publisher.bind("tcp://*:50000")
    workload_queriers = [context.socket(zmq.REQ)] * CAMERA_NUM
    for i, skt in enumerate(workload_queriers):
        skt.connect("tcp://192.168.3.{}".format(101+i))

    workloads = [0] * CAMERA_NUM
    while 1:
        for i, skt in enumerate(workload_queriers):
            skt.send_string('Requesting workload')
            message = skt.recv_json()
            util.store_wkld(workloads, message)

        instr = util.balancer(workloads)
        time.sleep(5)
        instr_publisher.send_json(instr)

    # never reach here
```

```
    instr_publisher.close()
    for skt in workload_queriers:
        skt.close()
    context.destroy()
```

E.4 Main part for client

```
import zmq, random, _thread
from . import util

CAMERA_ID = 0   # the id of the current camera

# thread for receiving message
def tran_recv(socket):
    while 1:
        message = socket.recv_string()
        print(message)
        socket.send_string("0")

def main():
    context = zmq.Context()
    workload_reporter = context.socket(zmq.REP)
    workload_reporter.bind("tcp://*:{})".format(50001+CAMERA_ID))
    instr_receiver = context.socket(zmq.SUB)
    instr_receiver.connect("tcp://192.168.3.201:50000")
    instr_receiver.subscribe("")
    recv_socket = context.socket(zmq.REP)
    recv_socket.bind(f"tcp://*:{40001 + CAMERA_ID}")
    send_socket = context.socket(zmq.REQ)

    # thread for receiving transmission
    _thread.start_new_thread(tran_recv, (recv_socket, ))

    while 1:
        assert workload_reporter.recv_string() == 'Hi'
        message = {'id': CAMERA_ID, 'data': util.get_workload()}
        workload_reporter.send_json(message)
        instr = instr_receiver.recv_json()
        util.send_skt(send_socket, instr)

    # never reach here
    workload_reporter.close()
```

```
    instr_receiver.close()
    recv_socket.close()
    send_socket.close()
    context.destroy()
```

E.5 Main part for Obtaining CPU info

```
def getCpu(request):
  if request.method == 'POST':
    receive_data = json.loads(request.body.decode())
    mode = receive_data['mode']
    if not mode:
      resp = {'errorcode': 100, 'detail': 'Need Mode'}
      return HttpResponse(json.dumps(resp), content_type="application/json"
    if mode == 'CPU':
      f = os.popen('top -b | head -50')
      str = f.read()
      cpu = re.search(r'Mem : (.*) total, (.*) free, (.*) used, \
        (.*) buff/cache', str, re.M | re.I)
      total = 0
      free = 0
      used = 0
      buff = 0
      if cpu:
        # print (cpu.group(0))
        total = cpu.group(1)
        free = cpu.group(2)
        used = cpu.group(3)
        buff = cpu.group(4)
      resp = {'errorcode':0, 'time':time.time(),'total':total, \
        'free':free,'used':used,'buff':buff}
      return HttpResponse(json.dumps(resp), content_type="application/json"
  return HttpResponse("This is GET. Use POST")
```

E.6 Main part for Visualization

```
const Camera1: React.FC = () => {
  function GetCpu(req) {
    const uri = `https://#ip_address/get_cpu_period`;
    const headers = { 'Content-Type': 'application/json' };
    const body = JSON.stringify(req);
    // const body = toParams(req)
    return fetch(uri, { method: 'POST', headers, body });
```

```
}
async function tryapi() {
  try {
    const res = await GetCpu({
      time_a: '#sample time 1',
      time_b: '#sample time 2',
    });
    const data = await res.json();
    const tmpTime = Array(0);
    const tmpUsed = Array(0);
    const tmpFree = Array(0);
    const tmpBuff = Array(0);
    const tmpTotal = Array(0);
    // console.log(data);
    data.cpu_period.map(item => {
      tmpTime.push(item.time.split('.')[0]);
      tmpUsed.push(item.cpu_used);
      tmpFree.push(item.cpu_free);
      tmpBuff.push(item.cpu_buff);
      tmpTotal.push(item.cpu_total);
    });
    setTime(tmpTime);
    setUsed(tmpUsed);
    setFree(tmpFree);
    setBuff(tmpBuff);
    setTotal(tmpTotal);
    // console.log(tmpTime);
    return data;
  } catch (err) {
    return err;
  }
}
useEffect(() => {
  tryapi();
}, []);
// const [value, setValue] = useState('');
const [time, setTime] = useState(Array(0));
const [used, setUsed] = useState(Array(0));
const [free, setFree] = useState(Array(0));
const [buff, setBuff] = useState(Array(0));
const [total, setTotal] = useState(Array(0));
const [selectFrame, setSelectFrame] = useState(1);
```

```
const [dataBeforeBalance, setDataBeforeBalance] = useState([
  {
    value: selectFrame === 1 ? 6 : 2,
    itemStyle: {
      color: '#990033',
    },
  },
  5,
  1,
  11,
]);
const [dataAfterBalance, setDataAfterBalance] = useState([
  {
    name: 'Before Balance',
    data: dataBeforeBalance,
    type: 'bar',
  },
  {
    name: 'Balanced',
    data: [
      {
        value: selectFrame === 1 ? 6 : 4,
        itemStyle: {
          color: '#ffcc00',
        },
      },
      5,
      selectFrame === 1 ? 6 : 5,
      selectFrame === 1 ? 6 : 5,
    ],
    type: 'bar',
  },
])

return (
  <div
    style={{
      position: 'absolute',
      overflowY: 'scroll',
      overflowX: 'hidden',
      height: 'calc(100% - 15px)',
      top: 0,
```

```
    width: 'calc(100% - 15px)',
  }}>
  <div style={{ height: 1700, backgroundColor: '#ffffff' }}>
    {/* <canvas
    style={{ height: '20%', backgroundColor: '#eeffff' }}
    onMouseDown={e => console.log(e.clientX)}
  /> */}
    <h1 style={{ position: 'relative', top: 20, left: 20 }}>Camera 1</h
    <Pagination
      total={200}
      style={{ padding: 20 }}
      onChange={v => {
        setSelectFrame(v);
      }}
    />
    <img
      style={{ padding: 20, width: '90%' }}
      src={imgData[selectFrame - 1]}
      alt=""
    />
    <h2 style={{ padding: 20 }}>
      Result will be later labeled on the graph{' '}
    </h2>
    <ReactEcharts
      style={{ padding: 20, height: 400 }}
      option={{
        title: {
          text: 'Workload at each frame',
        },
        legend: {},
        tooltip: {},
        dataset: {
          dimensions: ['product', 'Before Balance', 'Balanced'],
        },
        xAxis: {
          type: 'category',
          data: ['Camera 1', 'Camera 2', 'Camera 3', 'Camera 4'],
        },
        yAxis: {},
        series: [
          {
            name: 'Before Balance',
```

```
            data: dataBeforeBalance,
            type: 'bar',
          },
          {
            name: 'Balanced',
            data: dataAfterBalance,
            type: 'bar',
          },
        ],
      }}
    />
    <ReactEcharts
      style={{ padding: 20 }}
      option={{
        title: {
          text: 'CPU at each frame (KiB)',
        },
        tooltip: {
          trigger: 'axis',
        },
        legend: {
          data: ['cpu_used', 'cpu_total', 'cpu_free', 'cpu_buff'],
        },
        grid: {
          left: '3%',
          right: '4%',
          bottom: '3%',
          containLabel: true,
        },
        toolbox: {
          feature: {
            saveAsImage: {},
          },
        },
        xAxis: {
          type: 'category',
          boundaryGap: false,
          data: time,
        },
        yAxis: {
          type: 'value',
        },
```

```
            series: [
              {
                name: 'cpu_used',
                type: 'line',
                stack: '总量',
                data: used,
              },
              {
                name: 'cpu_total',
                type: 'line',
                stack: '总量',
                data: total,
              },
              {
                name: 'cpu_free',
                type: 'line',
                stack: '总量',
                data: free,
              },
              {
                name: 'cpu_buff',
                type: 'line',
                stack: '总量',
                data: buff,
              },
            ],
          }}
        />
      </div>
    </div>
  );
};
```

Acknowledgements

We thank Prof. Zhu Yifei for his meticulous instructions. Technical supports from Principle Engineer Song Danyang assisted the manufacturing. Informative lessons given by Prof. Wu Jigang inspire our ideas.

上海交通大学 毕业设计（学士学位论文）
单独工作报告

SHANGHAI JIAO TONG UNIVERSITY

CAPSTONE DESIGN (BACHELOR'S THESIS)

INDIVIDUAL CONTRIBUTION REPORT

Student Name：岑盛

Student ID Number：517370910045

Major: Electrical and Computer Engineering

I’m the team leader of the project. In our capstone design project, I was in responsible for the overall organization of each course activity, including the project progress supervision, design review presentation, report, final thesis, and poster design, etc.. Also, I was engaged in the overall physical architecture system design, the model and algorithm deployment in cameras, network setup for the whole system. I closely communicated with my team members and instructor every week, and reported encountered problems to the instructor regularly.

Here’s my teammates’ contribution of our project. Jin and Yang helped set up the network connection for the whole system, including the ssh connection among all the devices (cameras, exchanger, NVDIA edge device, and the edge processor), and the “opport” connection for the situation when experimenters are outside the lab. Pan was in responsible for the algorithm design part, including the workload balance algorithm, phase I object detection bounding box generation part, and phase II vehicle detection part. Besides, Jin explored to precompile the rknn model, and run the model on cameras. And Yang was also responsible for the simulation of message transmission between camera and camera, and between camera and edge device. Finally, Zheng was in charge of the front-end part. She retrieved raw image data, bounding box number data, bounding box transmission status, inference result,...., and helped visualize on the screen. This part is mainly for demonstration and experiment result analysis.

Now, let me describe my detailed technical tasks. First, with the assistance of Jin and Yang, I connected the cameras with the main edge processors on an exchanger, which receives network signal from a router. The ip address could be found using our local HUAWEI router. Second, I designed the overall architecture of the whole system. The system requires full-duplex communication between cameras and the edge device. The camera first deployed a model, which is used for bounding box detection and type classification based on YOLO, in a docker which was provided by our NPU company. In the docker, the environment can be used for NPU inference. Then part of

the phase I result (mainly the bounding box number) was transmitted to the edge device. The edge device implemented the workload balance algorithm to keep the bounding box number roughly the same among all cameras. The algorithm also outputs a matrix for each timestamp, which guides the transmission among cameras. Then the bounding boxes are transmitted between cameras, to feed into phase II model to recognize the object color and vehicle detailed type (if it's recognized as a vehicle). Third, I deployed rknn models in camera for phase I inference. Necessary environment setup is accomplished. Last, I assisted Jin to deal with the transformation from pytorch model to rknn model which NPU recognized.

After the whole process of our capstone design, I learned a lot of engineering tools and analysis methods. First, I learned the basic knowledge on docker, as well as has got the experience of building and implementing docker. Docker is a very powerful container that encapsulates the running environment and related codes. It can be implanted into various running machine, as long as the operating system supports. At beginning, I do not know what's the application of docker, and I do not know the basic function of it as well. I read many reference documents, and run some online examples to get acquainted with docker usage. Second, I gained experience in running deep learning models on a device with NPU drivers. The deep learning should be first converted to an rknn model, which is a standard form in our smart cameras. The rknn transferring code cannot be run directly on my host machine, neither on my virtual machine. Then I downloaded the docker image containing the function of rknn api, which could be implanted on Linux x86 system. Also, the docker image could not be implanted on our cameras, since our cameras are in aarch64 system. So with the help of the colleagues of our instructor, we got the docker version which could be run on our camera edge. Third, I learned file transmission scheme used in typically engineering field, although I'm not in charge of the communication part. The traditional ssh connection can be combined with ftp transmission guidance, embedded in the python file. Last but not least, I learned various engineering complexity, like the matching problem between different libraries and python, the compatibility between a

docker image and operating system (aarch64/x86).

From the experience in conducting our research in VE450, I encountered lots of technical problems. I collaborated with Jin to deal with the rknn transfer problem, the network connection problem. The solution to the rknn transfer problem is really tedious, both the docker methods and the raw rknn api installation methods did not work in the beginning. The former one did not work because of the platform incompatibility issue, while the latter one failed due to the conflicting of versions for different libraries for python. I collaborated with Pan to design the appropriate deep learning model for bounding box detection and type detection. The model not only should be relatively accurate, but also should be transferred to rknn file (fit into the requirement of library version match) smoothly and inferred on camera edge in an acceptable time period. I discussed with Zheng on what to show on front-end monitor. How to visualize the workload balance and bounding box better.

VE450 is really an unforgettable experience on guiding me how to communicate with my teammates and instructor more efficiently, how to analyze a problem more comprehensively, how to solve a technical problem more quickly, etc.. There still remains to be some problems we could not solve, but we have already found out many possible solutions to them. When meeting with problems, we need to make a choice on solving it by ourselves (like online reference searching), or by consulting some professionals (like technical staff in NPU company). Both are good ways, but we should try one first according to the complexity/type of the problems.

上海交通大学 毕业设计（学士学位论文）
单独工作报告

SHANGHAI JIAO TONG UNIVERSITY

CAPSTONE DESIGN (BACHELOR'S THESIS)

INDIVIDUAL CONTRIBUTION REPORT

Student Name：金哲健

Student ID Number：517370910167

Major: Electrical and Computer Engineering

In the capstone project of Cross-Camera Video Analytics at Edge, I was responsible for the back-end part and algorithm realization part. In the back-end part, I did the network setup (server and client configuration and implementation), cameras and edge side setup, also I designed the details of data transmission, and the realization of data transmission is finished by Yang and Cen. Pan designed the initial version of our algorithm including video analytics algorithm and workload balance algorithm, and I changed and realized the video analytics algorithm that adapted to the cameras by docker deployment, which includes model transfer and model inference. Cen is responsible for the algorithm realization on the NVIDIA edge side, and Zheng is responsible for the front-end part. Each of us had different individual jobs and we worked together to finish this project.

First I set up the physical connection of our systems. Multiple cameras embedded with NPUs are connected via a switch, which is connected to an upstream router for getting LAN. An NVIDIA Jetson Xavier is also connected through the switch for the purpose of workload balance. When setting up the network after setting up the cameras successfully, in order to access the cameras in LAN in the laboratory remotely, I decided to use a virtual private server for intranet penetration. This process provided great convenience for our remote development.

Pan designed the camera video analytics pipeline. For each video frame, the smart camera will first run a YOLO object detection model to extract bounding boxes from the frame. After receiving commands from the server, it will run different video inferences upon different types of assigned bounding boxes. If a bounding box displays a vehicle, it will run another YOLO model to detect its type. If the vehicle type is either car or truck, a CNN model will be deployed to identify its color. Under another circumstance, a bounding box shows a person, it will just run a CNN model to recognize its cloth color. If the bounding box displays other types of objects, a new YOLO model will be used to identify whether it is an explosive.

The most difficult and challenging thing is the model deployment in edge-side cameras. Traditional python code of deep neural networks is hard to be deployed in the edge-side cameras, due to the fact that our RK1808 camera is designed to run neural networks on NPUs. In our project, we need to first transfer the original deep learning model to the rknn model through a tool provided by Rockchip on x64 computers. Then, the transferred rknn model is encapsulated into the docker in aarch64 cameras, and then the cameras will run model inference using NPU. I’ve met with several difficulties in this process, for example, the python code provided by Pan that can run on her x64 windows cannot be run directly on the neural processing units on the cameras, it needs to be changed a lot to be compatible with the model transfer tool. Besides, there are problems related to version compatibility of the rknn transfer tool on x64 side and aarch64 side. Also, there are bugs that almost no one asks about online. The troubleshooting process is quite challenging, and I’ ve tested different deep learning models such as ssd, mobilenet, yolov2 and yolov5 on several original deep learning platforms such as pytorch, tensorflow, darknet and caffe. Fortunately, after communicating with engineers of Rockchip and Jiangxing Intelligencein time, I solved most of the problems.

The data transmission part can be divided into camera-server and cross-camera connection. After comparing different message queue tools such as ZeroMQ, RabbitMQ and ActiveMQ, and other data transmission tools such as Kafka and grpc, I decided to use grpc on the camera-server side and ZeroMQ on cross-camera side. However, we decided to use ZeroMQ on both of the two data transmission parts due to the implementation complexity. My teammate Yang Xuting realized the data transmission part on python code.

The front-end part is finished by Zheng. In this part we visualize results and quantify the degree of optimization. The pipeline here can be separated into 3 parts, obtaining CPU info, obtaining workload info, and visualization. All API for obtaining CPU info are written in Python using Django. In this part, in order to do the jobs concurrently,

Cen and I gave Zheng some test examples in advance. Besides, for expo convenience, I set up a computer connected by an extra Nvidia edge device with a monitor for visualization.

Our system is actually very challenging to implement. It’ s a complete end-to-end system, from the camera recognition, to the NPU computing, to the data communication, then to the front end realization. That is to say, this system can be put into commercial use directly, and it’s not a simple problem of neural network module reproduction. Most cameras on the market now have no computing functions. no NPU embedded. no docker isolation, no data communication among cameras, no complex chain structure on the side of multiple modules, and certainly no workload balancing among cameras.

The capstone project is a very good platform for me to use what I’ve learnt during my undergraduate period to design a real product. In the whole developing process, I’ ve learnt how to use a virtual private server for intranet penetration, how to use docker to package my development environment code profiles into the container and publish and apply it to any platform. I’ ve learnt how to implement a deep-learning system onto a NPU (neural processing unit) chip. I’ve also learnt how to implement ZeroMQ to do data transmission. In the process of development, we 5 teammates worked collaboratively and efficiently. We have designed a video analytics system that dynamically distributes workloads across smart cameras with the assistance of an edge cluster. We aim to lower the latency and avoid queue congestion by maximizing the utility of all the computing resources in the system compared to the benchmarks. I’ m very satisfied with our work and proud of us, since we have overcome so many challenges and difficulties all the way.

上海交通大学 毕业设计（学士学位论文）
单独工作报告

SHANGHAI JIAO TONG UNIVERSITY

CAPSTONE DESIGN (BACHELOR'S THESIS)

INDIVIDUAL CONTRIBUTION REPORT

Student Name：杨煦庭

Student ID Number：517370910013

Major: Electronic and Computer Engineering

As one of the members of group 22, my capstone design project is titled Cross-Camera Video Analytics at Edge instructed by Prof. Yifei Zhu. As a group of 5 people, each one has different backgrounds, as well as the strengths and weaknesses. To push the project to go under smooth procedure, we work together by taking up different parts and coordinate our achievements with regular meetings. In this paper, I will be summarizing my individual contribution to our design project and assess my acquisition from the procedure.

Capstone design projects in JI is not only a project, but a course as well, from which we are expected to learn not only engineering knowledge and skills but the abilities for technical communication. And I will be discussing the contributions in these two aspects.

My main responsibility is to design and construct the physical and logical network architecture for the system. As a video analytic system, multiple devices should be co-operating under one same local network, and proper methods are to conduct the devices, not only to realize the inter-device communications but boost the function of the system. As the physical basis, we connect assemble the system by connecting al the device to a local network support by a route and a switch. I bind each device to a static IP address so that IP-based access can be available. As a network deployed in lab, one major problem is that remote access is not possible. To solve this, together with my teammate Zhejian Jin, we deploy FRP, an open-source reverse proxy, on a remote server with public IP as well as on the local device, realizing a NAS traversal to support remote development. Moreover, I setup the proxy as a system service to ensure more stable functioning, resistant to possible power or network problems.

Besides, I design and implemented the network structure for the system. One of the jobs for the edge cluster device is to requesting the workload data from all the cameras in the system and hand out instruction, and therefore we need a server-client structure here. Onn the other hand, since each there will be data transmission between any pair of cameras, all of them are of the same status, thus we need a P2P network. To implement the communication, I choose ZeroMQ, an open-source message query

library with high efficiency, low cost as well as good language support. Without rich knowledge in computer networks, I have to learn basic protocols and common operations by wide search on the Internet. After reading documentations for ZeroMQ for many times, I came to understand the principles and master usage of this library, and try to write some example programs. In the end, I implemented the basic functions of ZeroMQ-based clients and servers in Python language for our project, which are later integrated with the program by my teammates to fulfill our need for network communication.

In additional to technical part, literature part is as well important for the capstone project. Instead of highly-technical descriptions of the topics, our expression should be friendly to non-technical people. This adds quite a few difficulties to our project presentation and the reports. In the composition of the slides, I came to realize the importance of a good structure, which is the key for audience to understand you. Keeping the syllabus tidy and clear makes it easy for the minds of the audience to follow. Besides, we need to come up with all kinds of methods to facilitate our description. Good slide typesetting structure help audience grasp the topics visually. Proper insertion of images or animations make complicated principles intuitive. Most of these skill are mentioned in the course before, but the capstone project offer plenty of chances to put them into application.

My largest literature contribution is making the promotion video. Limited to 3 minutes, the video needs to show our project completely and clearly. After designing and revising the syllabus for many times, I manage to concentrate everything from background, problem, to implementation and validation in the required time period. Fortunately, with rich experience in video processing, I collected resources, design storyboard and add subtitles efficiently. The video would work as the signboard for our project in front the large design expo audience.

This design project offered me plenty of experience and feelings. My very first feeling is how important teamwork is in an engineering design project. Dislike a naïve understanding of teamwork, that everyone works on something together, I realize that the teamwork is the proper coordination between different work done by different

people. In our project, almost every member worked on different parts of the design, like algorithm, analysis pipeline, network, model deployment and so on. And we communicate with each other through regular meetings and decide how we can help each other to push the project running. Since one cannot fully understand the contents done by others, the only thing possible to do is closely follow the schedule or plans. Therefore, in addition to many people working together on one project, a good workload conduct is essential to keep the procedure run as planned.

The VE450 capstone also taught me of how engineer design works. The specific steps from literature review at the beginning, to generating concepts and develop detailed designs, with reports and presentations at different stages, the process of engineering design became clearer and more formal to me. In the past, I only regard engineering as a technical job, where in most of the time people deal with the project itself. However, this makes the process lack of plan and tidiness. The object may not be clear, the design would keep on changing, and the job of different members become hard to coordinate. This reminds me of the importance of literature work during a design project, and this course offer me more insight in the future career in the field.

From this project, my ability to generate answers was greatly trained. Our project, as mentioned above, involves contents from many fields. It is almost impossible for someone to have a mastery of all those knowledges at the same time. Individually, I read plenty of essays and technical articles to learn basics in this fields, including networks, computer vision, Linux-based development and so on. Another important way to learn is by communication, which is also part of teamwork as narrated before. Since each member has strongness in different parts, by exchanging our minds, we can learn efficiently. I really appreciate my teammates for the help in those times.

Above all, I feel fortunate to be able to in the capstone design project, of cross-camera video analytics at edge. From my experiences in designing and implementing system network architecture as well as the literature jobs, I grasp the meaning of teamwork and have a better understanding about engineering. I have a more positive view in the study and career in this field, and this capstone project

would remain an impressive experience in my life.

上海交通大学 毕业设计（学士学位论文）
单独工作报告

SHANGHAI JIAO TONG UNIVERSITY

CAPSTONE DESIGN (BACHELOR'S THESIS)

INDIVIDUAL CONTRIBUTION REPORT

Student Name：潘启颖

Student ID Number：517370910039

Major: Electrical and Computer Engineering

We distributed our workload evenly. I have taken part of the algorithm design in this project. Cen Sheng is responsible for architecture design. Yang Xuting and Jin Zhejian write the code for back-end programs. Zheng Zhirui designs the front-end program.

We need to design an algorithm that can balance the different workloads among cameras. The algorithm should lower the variance of workloads larger than 30%. However, the algorithm cannot be time-consuming as we want to process as many video frames as possible per unit time. I come up with three concepts to resolve the problem. First, we can model the problem as an optimization problem and solve it using mathematical tools. However, the formulated optimization problem is nonlinear as we need to take many practical issues into account. For example, if the variation of workload is quite small, there is no need to balance at all. If balanced, the system spends more time than not balanced as data transmission consumes time as well. Next, this is a policy-making problem. Reinforcement learning can be deployed. The advantage of RL is that with learning time increasing, the action the system takes tends to optimize overall rewards. However, the determination of best action takes much larger time than expected. We cannot use this method. Last but not least, a heuristic algorithm can be utilized. It is easy to implement and fast to run. The effectiveness can also be ensured.

When designing the heuristic algorithm, the problem I encounter is how to quantify the practical considerations. Workload overheading is a situation to prevent from. So I introduce a triggering threshold and an equation to measure the balancing condition of the whole system. Also, after determining a migration pair to transfer workloads, the exact number of workloads to transmit is hard to compute. Rather than a constant number, I decide the number is proportional to the current workload of the camera to transfer photos from. The ratio is dependent upon different scenarios and decided by trial and error.

The balancing algorithm needs experiments to validate its function. The first problem I encountered is to select a proper dataset. The dataset needs to be taken from multiple

cameras simultaneously for a period of time. In addition, the camera locations should vary from one to another. Otherwise, the functionality of our system cannot be demonstrated. After looking up the Internet video analytical resources, we decide to use VIRAT dataset for demonstration.

The next task to accomplish is to design a complete work flow for a single edge camera. A smart camera can not only detect objects but also can further run inferences based on the type of objects. The work flow is determined by the dataset. I recognize that in the video cameras, there are different types and colors of vehicles and people dressed in different colors. In real life surveillance video analytic system pays much attention upon the type and the color of vehicles and people cloth colors to identify suspicious people or cars. Then we need to implement some algorithms to finish these different tasks. Luckily online github resporitories provide many well-designed algorithms. Hence, I clone these resporitories, test their efficiency with our datasets and choose the final algorithms.

When generating different concepts for balancing algorithm, I learn how to formulate a realistic problem using mathematical models. I also learn how to compare the efficiency of different methods and determine the final solution. A survey is conducted to know the pros and cons of the three methods. From the survey, I know that solving an optimization problem will obtain a result with the best accuracy. However, how to solve it remains a problem if the objective function and constraints are non-convex. In practice this problem is very difficult to solve. Reinforcement learning is widely-used in various fields, which proves its great potential. But people usually deploy RL when the time limit is not constraint and the computing resource is rich. With more training rounds, RL acts more wisely. A heuristic algorithm, though providing results with the least accuracy, it is simple to implement and fast to complete. Sometimes, engineers use heuristic algorithms due to hardware constraint or other practical concerns.

The process of problem formulation gives me an idea of quantization. Weight factor is of great use during quantization. Under many circumstances, the objects, though of same type, feature heterogeneously. For instance, in real life, the smart cameras may

be manufactured from different factories and their computing capacity are different. To take this into account, we need to use the weight factor to modify the index for an individual camera. \

A search for an appropriate test dataset sounds easy, but takes a lot of time. Just typing the requirements for the dataset in the search engine cannot resolve the problem. I find that one of the most convenient way is to read related articles or papers. Article writes introduce the datasets they use in details and provide direct access to them.

The implementation of work pipeline in a single camera teaches me how to use pytorch, tensorflow and caffe to build neural networks. Thanks to the excellent adaptaions of rknn models, I can clone network models written with different packages. Many video analytical inferences have pretrained models online. Therefore, cloning them is the fastest way to complete the workflow. Pytorch and tensorflow provides multiple model saving methods. In different environment different model types are needed.

In general, I succeed in designing a workload balancer algorithm and in testing the algorithm by building a workload pipeline dependent on a specific dataset. The whole process gives me an idea of designing and implementing a video analytic system, which is very popular nowadays. I also learn how to communicate with my group members as my part need to interact with their parts. What’ s more, I learn how to communicate with sponsor to find out his needs. The technical communication lessons in VE450 teach me how to present my ideas and my work to the audience in a professional way.

上海交通大学 毕业设计（学士学位论文）
单独工作报告

SHANGHAI JIAO TONG UNIVERSITY

CAPSTONE DESIGN (BACHELOR'S THESIS)

INDIVIDUAL CONTRIBUTION REPORT

Student Name：郑智睿 Zheng ZhiRui

Student ID Number：517370910044

Major: ECE

Our capstone project is Cross-Camera Video Analytics at Edge, and we are a group of five. We had meetings with our instructor once a week, and we worked together for the task. Current problems with cloud-edge camera pipeline includes: For cloud only solutions, there may be high latency and constrained bandwidth, since huge size of information will need to be send. For edge only solutions, the computational resources are limited. So we would like to focus on the cooperation between cloud, edge, and cameras.

Ceng Sheng is responsible for the overall architecture design, and also helped with data transmitting. Pan QiYing chose the algorithm for balancing workload, and models for detection. Yang Xuting and Jin ZheJian are mostly responsible for hardware and back-end parts, along with network and data transmitting, and I designed front-end part and also some backend interfaces related.

We witness that the workload among different cameras differ a lot. If the workload is balanced, the overall processing rate may be raised. So we would like to come up with a design such that it can fully utilizes the resources of edge and cloud, partition workload properly between cloud and edge, achieving good tradeoff between network delay and resource usage, allocate workload dynamically among cameras.

Our task in this project is to balance workload among different cameras dynamically. Therefore, algorithm is designed for partition workload between edge, cloud and cameras, so that latency and bandwidth consumption can be balanced. We have to show our results in the expo, so the results need to be visualized so corresponding front-end is needed, because showing raw data and codes would not be friendly to viewers, visualizing plays quite an important role.

As we begun, we each read essays and studied benchmark products and gathered information. We shared our throughts during our weekly meeting, and Ceng Sheng and Pan QiYing came up with the first architecture design of our project. For hardware part, physical connection is set by Jin ZheJian. The NVIDIA Jetson Xavier is connected through the switch along with all the cameras. Therefore, my interfaces

can get information from both phase without further communicating with the cameras separately. Pan then designed the camera analytics pipeline. For each frame, a YOLO object detection model will run on each camera separately (denoted as phase 1). Then the detected bounding boxes will be sent to the server. The server balances the workloads for each camera, and then the cameras will then run phase 2 to tell more information such as what color is the vehicle. As the server deals with workload and gets the final results, a copy of bounding boxes and workload is also sent to the interface severing for front-end purposes.

I am mainly responsible for visualizing results and quantify the degree of optimization. Original design would contain a frame by frame video that tics every 5 second and its corresponding workload. The design is later changed to show workload trend and source, and also containing info such as workload variance, throughput, time consumption, etc.

Ideal output will be a page showing original video by frame and the corresponding results, workload before and after balancing, and info such as CPU usage and time to prove the degree of optimization. In order to accomplish that, I first wrote several APIs to obtain and store the obtained info, including: The first API gets current memory (etc.) information and send it to the server with timestamp. Current timestamp and information needed is obtained by instructions, then the information needed can be separated through regular matching and sent through API to server side. The API is triggered per minute or second. The server then stores the data into MySQL database. Another API then call the memory (etc.) information data from certain period of time from the database. All API are written using Python Django. Then workload for each frame is stored in similar way. After the data for each frame is stored, the front-end can then display them frame by frame. Every 5 seconds I will call the interface to get the corresponding picture, result, and workload data, and the page will be re-rendered, showing results and workload dynamically. One problem here is that, for example, phase 1 will only give me type and range of each bounding box, and its color or other information I have to get through phase 2, i.e., another API. Therefore I wrote Python script to merge results and then directly send the processed

picture to the front-end side to save time and memory consumption. The front-end part is written using React Hooks. The reason for choosing React Hooks is that it can extract stateful logic from a component and reuse stateful logic without changing component hierarchy. The final design works as planned: the user can change views between each cameras, and for each page the user can see each frame of each camera, its corresponding workload before and after balancing, how the workload is balanced, and CPU usage at each frame. I choose Python and MySQL because I am familiar with them, for realizing this part, I searched for usage of React E-Charts, therefore most front-end part is done using React E-Charts. I also learnt the usage of hooks. Because I already had working experience with React, I can concur most problems I meet with. Future improving may include optimizing coding style, and encapsulating components and interfaces to make the codes more readable (since the codes are now quite huge).

In conclusion, I did well in my part of the project, and I succeeded in building a front-end page showing workload trend and processed results. I am confident that I would offer a good exhibition experience to expo viewers. The process of developing the project gives me a new idea about improvements that can be made through collaborative communication, and I also learned more communication skills among team members. I also learned a lot through tech communication lectures such as how to make my speech clear and show results to non-tech audience.