DISTRIBUTED MESSAGING SYSTEM FOR THE IOT EDGE

by

Anjus George

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Electrical Engineering

Charlotte

2020

ABSTRACT

ANJUS GEORGE. Distributed Messaging System for the IoT Edge. (Under the direction of DR. ARUN RAVINDRAN)

Recent advances in Deep Learning have made possible distributed multi-camera IoT vision analytics targeted at a variety of surveillance applications involving automated real-time analysis of events from multiple video perspectives. However, the latency sensitive nature of these applications necessitates computing at the Edge of the network, close to the cameras. The required Edge computing infrastructure is necessarily distributed, with Cloud like capabilities such as fault tolerance, scalability, multi-application tenancy, and security, while functioning at the unique operating environment of the Edge. Characteristics of the Edge include, highly heterogeneous hardware platforms with limited computational resources, variable latency wireless networks, and minimal physical security. We postulate that a distributed publish-subscribe messaging system with storage capabilities is the right abstraction layer needed for multi-camera vision Edge analytics.

We propose Mez - a publish-subscribe messaging system for latency sensitive multi-camera machine vision at the IoT Edge. Unlike existing messaging systems, Mez allows applications to specify latency, and application accuracy bounds. Mez implements a network latency controller that dynamically adjusts the video frame quality to satisfy latency, and application accuracy requirements. Additionally, the design of Mez utilizes application domain specific features to provide low latency operations.

In this dissertation, we show how approximate computation techniques can be used to design the latency controller in Mez. We also present the design of Mez by describing its API, data model and architecture. Additionally, Mez incorporates an in-memory log based storage that takes advantage of specific features of machine vision applications to implement low latency operations. We also discuss the fault tolerance capabilities of the Mez design.

Experimental evaluation on an IoT Edge testbed with a pedestrian detection machine vision application indicates that Mez is able to tolerate latency variations of up to 10x with a worst-case reduction of 4.2% in the application inference accuracy. Further we investigated two approximate computing based algorithms - a heuristic based pruning algorithm and a Categorical boost machine learning model based algorithm, to make the Mez's latency controller design scalable. Both algorithms were able to achieve video frame size reduction upto 71.3% while attaining an inference accuracy of 80.9% of that of the unmodified video frames.

ACKNOWLEDGEMENTS

Throughout the PhD program, I have received a great deal of support and assistance. I owe my deepest gratitude to my advisor, Dr. Ravindran whose expertise was invaluable in formulating the research topic and methodology in particular. He has constantly pushed me to be a better researcher, better engineer, and better writer, significantly broadening the scope of my abilities and contributions.

I would like to express my sincere gratitude to Dr. Tabkhi for providing valuable suggestions throughout my research work. I would like to thank him for allowing me to make use of the facilities in TeCSAR laboratory throughout my research work.

I am deeply indebted to my doctoral committee members for giving valuable suggestions that helped me to advance in my research. I would like to thank them for their time and consideration for being available to evaluate me in my major doctoral dissertation milestones.

Thanks to my fellow labmates in TeCSAR lab, for their wonderful collaboration. You supported me greatly and were always willing to help me. I appreciate your excellent cooperation and all the influencing ideas that helped me to progress in my research work.

I would like to sincerely thank the ECE department of UNC Charlotte, for providing me the opportunity to pursue my PhD. Thank you for assisting me financially with GASP award and allowing me to use the lab and research facilities in the department. I would also like to acknowledge the National Science Foundation for providing me research funding through the Smart and Connected Communities program.

Last but not least, I would like to thank my family for their unconditional trust, timely encouragement and endless patience. You are always there for me. Many thanks to my friends who have always been a major source of support, as well as providing happy distraction to rest my mind outside of my research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AC	Approximate Computing.
AMQP	Advanced Message Queuing Protocol.
AP	Average Precision.
API	Application Program Interface.
ARM	Advanced RISC Machine.
BLE	Bluetooth Low Energy.
CIFAR	Canadian Institute For Advanced Research.
COCO	Common Objects in Context.
CPU	Central Processing Unit.
CRC	Cyclic Redundancy Check.
DRAM	Dynamic Random Access Memory.
DSD	Directed Search Domain.
FN	False Negative.
FP	False Positive.
FPS	Frames Per Second.
GDBT	Gradient Boosting Decision Tree.
GPU	Graphics Processing Unit.
HAS	HTTP Adaptive Streaming.
HLS	Hue Lightness Saturation.

HSV	Hue Saturation Value.
HTTP	HyperText Transfer Protocol.
IoT	Internet of Things.
IoU	Intersection over Union.
JAAD	Joint Attention in Autonomous Driving.
LHS	Latin Hypercube Sampling.
mAP	Mean Average Precision.
ML	Machine Learning.
MOO	Multi-Objective Optimization.
MQTT	Message Queuing Telemetry Transport.
MTMC	Multi-Target Multi-Camera.
NBI	Normal Boundary Intersection.
NSGA	Non-dominated Sorting Genetic Algorithm.
PI	Proportional Integral.
PTP	Precision Time Protocol.
QoE	Quality of Experience.
RBAC	Role Based Access Control.
REVAMP^{2}T	Real-time Edge Video Analytics for Multi-camera Privacy-aware Pedestrian Tracking.
RMSE	Root Mean Square Error.

RPC	Remote Procedure Call.
SPEA	Strength Pareto Evolutionary Algorithm.
SPO	Successive Pareto Optimization.
SSD	Solid State Drive.
SSTable	Sorted Strings Table.
STOMP	Streaming Text Oriented Messaging Protocol.
TCP	Transmission Control Protocol.
TeCSAR	Transformative Computer Systems and Architecture Research Lab.
TLS	Transport Layer Security.
TP	True Positive.
TPM	Trusted Platform Module.
UDP	User Datagram Protocol.
USB	Universal Serial Bus.
Wi-Fi	Wireless Fidelity.
XGBoost	eXtreme Gradient Boosting.

CHAPTER 1: INTRODUCTION

The recent emergence of powerful machine vision algorithms based on Deep Learning has made possible Internet-of-Things (IoT) applications that utilize machine vision for a variety of challenging tasks including autonomous driving, pedestrian safety, public security, and occupational health and safety. Such applications involve computationally intensive processing of streaming videos from cameras operating 24x7x365. Assuming a modest frame rate of 5 fps, and a 500 KB frame size, 216 GB of data is generated per camera per day (19.5 Mbps per camera). Often, multiple cameras are needed to provide adequate area coverage for subject tracking and overcoming occlusions [1, 2]. Additionally, the aforementioned applications tend to be latency sensitive - that is, the processing of video frames needs to be done within a short time window for the results to be useful. The duration of the time window depends on the speed of the event (for example, tracking a high speed vehicle vs. tracking a pedestrian), and the response time needed for useful actions (for example, sounding an alert before vs. after event). Furthermore, while many of these applications enable broader smart city initiatives, and social good, significant privacy concerns exist regarding the potential misuse of the collected video data [3].

Despite the considerable computing power available in the Cloud, the use of Cloud computing for IoT machine vision applications is hindered by the high network latency to access a remote data center (typically hundreds of milliseconds)[4], and constraints in the upload bandwidth (typically tens of Mbps). Moreover, privacy and legal concerns place limitations on sharing of sensitive data to remote servers controlled by external entities. IoT machine vision applications are thus an ideal candidate for the Edge computing paradigm [5, 6, 7, 8, 9] , where most of the processing of the video streams happen in the vicinity of the camera. The Cloud may still play a role in aggregating detected events from multiple IoT Edge deployments, both for performing batch analytics, and for archival purposes. The localized processing of video streams at the Edge potentially allows for low latency operation, overcomes bandwidth limitations by reducing the size of the data that needs to be sent to the Cloud, and helps addresses privacy and legal concerns by eliminating the need to share raw video frames with Cloud vendors.

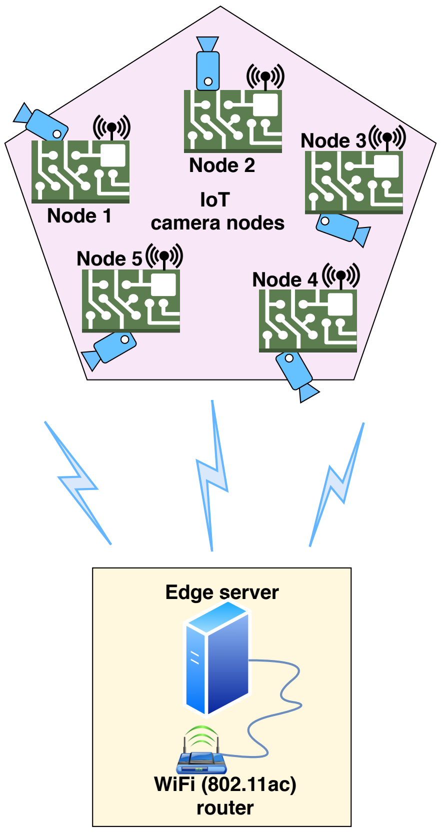


Figure 1.1: System architecture for machine vision at the IoT Edge. IoT nodes equipped with cameras record and transmit video frames to an Edge server through a Wi-Fi (802.11ac) wireless router. The Edge server runs machine vision algorithms on the received video frames for object detection, tracking and event prediction.

Figure 1.1 shows the system architecture for machine vision at the IoT Edge. Low power IoT nodes equipped with cameras, stream live videos of the area under observation to an Edge server equipped with GPUs through a wireless network. The

Edge server aggregates the individual video streams from multiple cameras, and run machine vision applications for object detection, tracking, and event prediction. For an application such as pedestrian safety (see Figure 1.2), the cameras are mounted on traffic signal posts at street intersections, and the Edge server is housed in a traffic signal box.

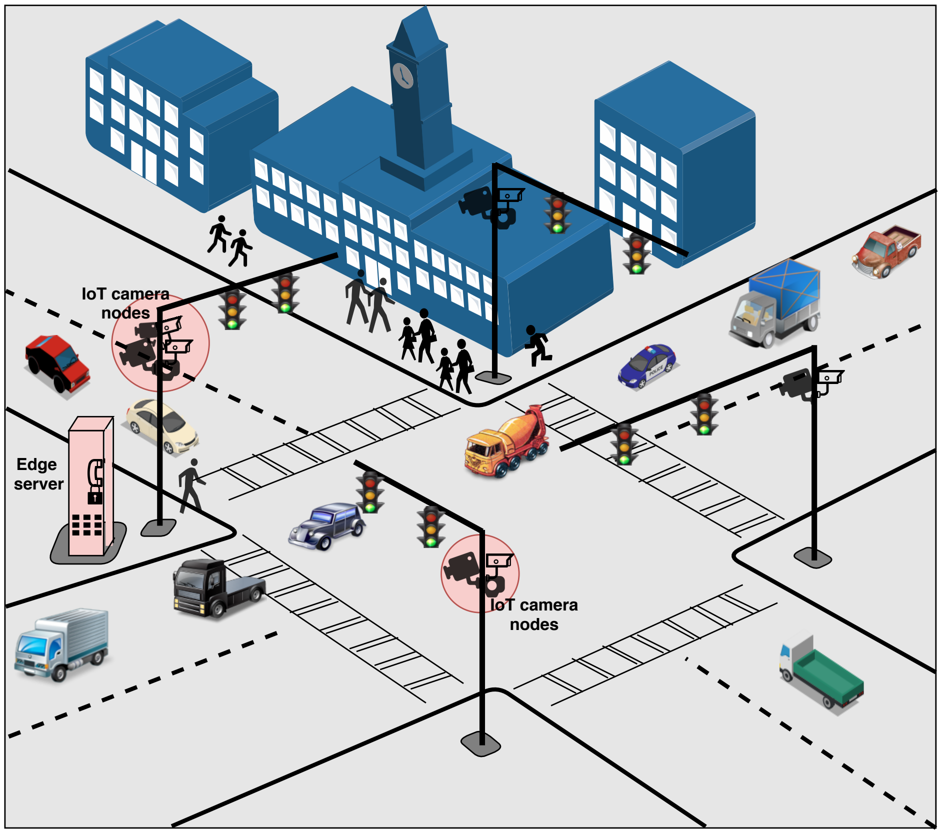


Figure 1.2: Multi-camera machine vision at the IoT Edge for pedestrian safety.

The Edge differs from the Cloud in some significant ways - unlike high speed wired networks in data centers, wireless networks at the Edge allow flexible installation of cameras at a lower cost. Also, cost reasons motivate the use of wireless technologies such as Wi-Fi (802.11ac) and Bluetooth (BLE) that operate in the free unlicensed bands. In contrast 4G and 5G wireless technologies operate in the licensed bands, and requires paid subscriptions to cellular network vendors. Additionally, space, cost, and power supply constraints limit the hardware redundancy available at the Edge.

Moreover, ensuring the physical security of the hardware is potentially challenging at the Edge due to deployments in unsecured environments.

In this dissertation, we explore the characteristics of an IoT Edge middleware layer that provides a suitable abstraction for machine vision application developers to deploy vision applications that consume video streams from one or more cameras. Since the applications are latency sensitive, the middleware layer should provide a means for applications to specify the latency requirements. The middleware then makes the best-effort to guarantee the specified latency. The use of wireless technologies such as Wi-Fi makes this particularly challenging, due to the large latency variations in the wireless channel. We propose the use of publish-subscribe (pub-sub) messaging system with storage, as a candidate IoT Edge middleware.

A pub-sub system decouples publishers (cameras) from subscribers (machine vision applications). IoT camera nodes publish video frames to topics identified by a camera ID. Applications subscribe to one or more topics as needed. The storage layer allows temporal decoupling of publishers and subscribers, allowing subscriber applications to access past video frames. In Cloud computing, such pub-sub systems are widely deployed to handle real-time data feeds, and as message brokers between microservices. Open source examples of such messaging systems include Kafka [10], NATS [11], and RabbitMQ [12]. Kafka is designed for high throughput, NATS targets low latency, and RabbitMQ allows for complex routing between publishers and subscribers. However, the existing messaging systems are built for the Cloud, where machines communicate over low latency wired networks (Gigabit Ethernet, Infiniband), and as such do not provide mechanisms to guarantee latency when operating in wireless channels with large latency variations.

We introduce Mez, a pub-sub messaging system specifically designed for machine vision applications at the IoT Edge. The key design themes of Mez are -

- Approximate computing - Mez exploits the trade off between video frame transfer latency from the IoT camera node to the Edge server, and video frame quality (approximate computing) inherent in machine vision applications. A lower quality video frame has a smaller size, and hence can be transferred with lower network latency. If a lower quality frame can provide acceptable accuracy, then a lower quality frame could be transferred from the IoT camera node to the Edge server during conditions of high channel interference. It should be noted that the application developer has to determine acceptable latency-accuracy tradeoffs. Mez provides a means for applications to specify the upper bound on the latency, and the lower bound on the accuracy requirements. The accuracy in turn translates to the quality of the video frame that Mez has to deliver.

- Adaptive computing - Mez constantly monitors the operating conditions of the wireless channel. When channel interference is high, Mez automatically adapts the quality of the video frames such that both the frame transfer latency and accuracy specifications are met. Automatic adaptation of video frame quality in Mez is facilitated by a network latency controller.

- Domain specific design - Mez employs an on-demand video frame transfer from the IoT camera node to the Edge server to minimize wireless channel interference. Furthermore, Mez uses an in-memory log based storage that exploits application domain specific characteristics to implement a simple low latency storage.

We have implemented Mez on an IoT Edge testbed with multiple IoT camera nodes, and a single GPU equipped Edge server. The source code is available from our GitHub repository[1]. Experimental results indicate that for pedestrian detection application with the OpenPose multi-person 2D pose detection benchmark [13, 14, 15, 16], Mez is able to achieve the target latency in the presence of up to 10x increase in channel

[1]`https://github.com/Ann-Geo/Mez`

interference with an application accuracy degradation of at most 4.2%. In contrast, the state-of-the-art NATS messaging framework suffers from latency degradation as the number of IoT camera nodes scale.

We then revisit the latency controller in Mez and propose a scalable design for the controller using two approximate computing based techniques - a pruning heuristic based approach and a machine learning model based approach. We evaluated the approximate computing based techniques, on an object detection application using publicly available vision dataset. Experimental results indicate that for object detection vision application based on EfficientNet [17, 18] Deep Learning architecture on the Microsoft COCO 2017 [19] dataset, we obtained an average video size reduction of 71.33% with an inference accuracy of 80.93% of that of the unmodified video frames.

1.1 Contributions

This primary contributions of this dissertation are as follows:

1. **Approximate computing technique to exploit latency-accuracy trade-off experienced by machine vision applications**

   We exploited approximate computing techniques to trade off quality of video frames in order to achieve guaranteed frame transfer latency in the wireless channel. The resulting video frame quality directly impacts the inference accuracy of machine vision applications consuming these frames. We conducted a study of video frame quality vs. accuracy degradation for two Deep Learning machine vision applications involving pedestrian detection and object detection.

2. **Multiple video frame quality modification techniques**

   There are several ways in which video frames can be modified to constitute a reduced frame size in order to achieve a reduced frame transmission latency. We call these modifications as tuning knobs for video frames. In this dissertation we identified various unique ways in which video frame content can be modified.

3. **Network latency controller**

   We designed an approximate computing based latency controller that uses multiple video frame quality modification techniques to simultaneously satisfy network latency and inference accuracy requirements from machine vision applications by monitoring the operating conditions in the wireless channel.

4. **Pub-sub messaging system for the IoT Edge**

   We introduced Mez - a publish-subscribe messaging system for machine vision applications deployed at the IoT Edge. Mez allows applications to specify latency, and application accuracy requirements and incorporates a network latency controller to satisfy these requirements. Additionally, the design of Mez utilizes application domain specific features to provide low latency operations.

5. **In-memory storage for pub-sub message broker**

   We designed a low latency in-memory storage that can be integrated in pub-sub messaging system brokers to store and retrieve visual data recorded by multiple publishers. The storage also has persistence for achieving fault tolerance against temporary messaging system failures.

6. **Scalable network latency controller**

   We presented the design of two approximate computing based algorithms - a heuristic based pruning algorithm and a Categorical boost machine learning model based algorithm, to make Mez's network latency controller design scalable.

1.2 Dissertation Outline

Chapter 2 provides a brief overview on Edge computing, approximate computing and distributed messaging frameworks. Chapter 3 presents experimental characterization of Wi-Fi latency at the Edge, and design and evaluation of the network latency controller in Mez. Chapter 4 describes Mez API, internal architecture, and detailed design including brokers, in-memory log, and fault tolerance aspects. Chapter 5 presents the experimental evaluation of Mez on an IoT Edge test bed. Chapter 6 describes and evaluates the techniques proposed for scaling the network latency controller in Mez. Chapter 7 discusses different design decisions made in Mez, and concludes the dissertation with suggestions for future work.

CHAPTER 2: BACKGROUND AND RELATED WORK

In this Chapter, we first define Edge computing and present several challenges experienced by Edge computing systems. In the following Sections, we provide an overview about approximate computing, computer vision applications, messaging systems, control systems and machine learning in the context of this dissertation. We also review the state-of-the-art work done in Edge computing, approximate computing and distributed messaging systems in their relevant Sections.

2.1 Edge Computing

Edge is defined as any computing or network resource along the path between data sources and Cloud data centers. In the Edge Computing paradigm compute and storage resources are placed at the Edge of the network close to data sources [20]. This is in contrast with Cloud computing paradigm where the data is cached and processed in Cloud data centers.

Architecture of a three tier Edge computing model is shown in Figure 2.1. The three tiers in this architecture are IoT, Edge and Cloud. First tier is IoT including smart home appliances, smart health wearable devices, cameras, smart grids and connected vehicles. Second tier, Edge consists of Edge servers (workstations) with more compute and storage capabilities than IoT devices. IoT devices communicate to the Edge servers using wireless technologies like 4G/5G cellular networks, Wi-Fi or Bluetooth. The third tier is composed of Cloud data centers (servers) with huge compute and storage capabilities. Some of the major Cloud service providers are Amazon Web Services [21], Microsoft Azure [22] and Google Cloud [23]. Communication between Edge and Cloud is facilitated by large throughput and high speed protocols like

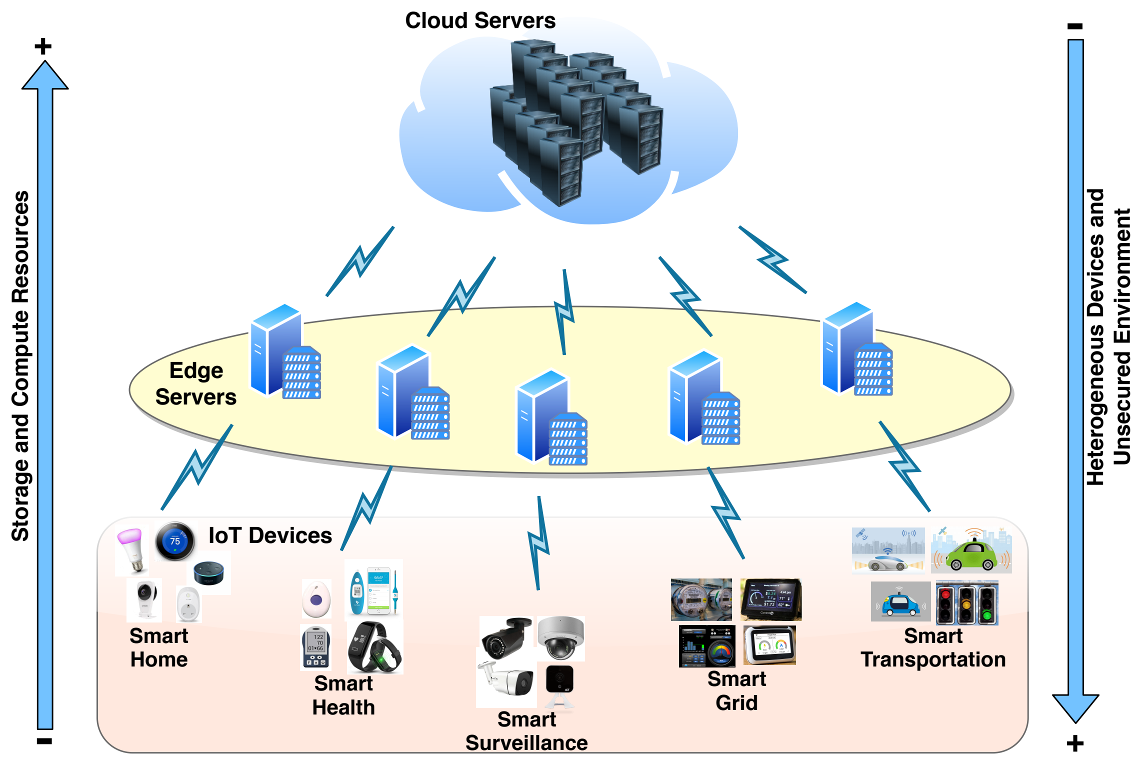


Figure 2.1: Architecture of a three tier Edge computing model. Tier 1 - consists of IoT devices and smart appliances. Tier 2 - includes Edge servers with more compute and storage capacity than devices in tier 1. Tier 3 - composed of Cloud data centers with huge compute and storage capacity.

Ethernet or optical fibers.

In Cloud computing paradigm, most of the computations happen in Cloud data centers resulting in longer latency as contrasted with the Edge computing paradigm where substantial amount of data is cached at the Edge for processing, contributing to shorter latency. However, there are several factors that make Edge a very challenging environment compared to Cloud.

- Cloud systems are housed in pristine data centers. On the other hand, Edge systems are often deployed in the "field" where highly dynamic operating conditions exist. For example, the Edge nodes most likely use a wireless communication link operating in unlicensed bands, where significant intermittent interference exists from other users.

- The data is inherently distributed at the Edge nodes due to the distributed nature of the data sources (for example, cameras). This is different from the Cloud where the data is distributed by design across multiple nodes to accommodate large data sizes.

- The heterogeneity of the storage nodes at the Edge is far more diverse than the Cloud. From a storage perspective, the embedded boards (IoT devices) have GB of storage, the Edge servers (at access points/base stations) offer TB of storage, and the backend Cloud offers PB of storage.

- The physical insecurity of the nodes in the field and the use of the Edge in critical cyber physical systems bring additional security challenges to the Edge [24].

In this dissertation we built an Edge Computing system similar to the one shown in Figure 1.1. Our system consists of multiple IoT cameras and an Edge server and is used to deploy computer vision applications (described in Section 2.3). The remaining part of this Section describes prior work done on vision based IoT Edge systems.

The concept and motivation behind Edge computing are described in a number of recent publications [5, 6, 7, 8, 9, 25, 26, 27, 28]. Regarding machine vision at the Edge, in the Gabriel project [29], Ha et al. describe a wearable cognitive assistance system where the images captured by a mobile device are processed by the Edge node to analyze what the user is seeing, and provide the user with cues as to what is in the scene (for example, recognizing a person). In the VisFlow project, Lu et al. [30] describe a system that can analyze feeds from multiple cameras for license plate recognition and real-time traffic flow mapping. In [31], Neff et al. proposes REVAMP^{2}T, an IoT system that tracks pedestrians across multiple cameras by running custom-designed deep learning based vision engines at the low power Edge nodes close to the cameras. However, none of these works address guaranteeing of latency requirements at the

Edge for machine vision applications.

In the Hetero-Edge project, Zhang et al. [32] describe a system that can efficiently orchestrate real-time vision applications on heterogeneous Edge servers. The new resource orchestration platform developed, uses a set of task scheduling schemes to make the Hetero-Edge system latency-aware, but does not consider communication latency. In [33], Pakha et al. introduce the idea of control knobs such as frame selection and area cropping to parametrize a custom video protocol that streams videos from cameras to Cloud servers to perform neural-network-based video analytics. Their work highlights opportunities to improve the trade-offs between bandwidth usage and inference accuracy, but does not address Edge specific latency requirements demanded by many IoT vision applications. In [34] Canel et al. proposes a new edge-to-cloud system called FilterForward that backhauls only relevant video frames from cameras to datacenter applications with the help of lightweight edge filters. However, unlike our approach, they perform computationally expensive Deep neural network based object detection at the camera nodes.

2.2 Approximate Computing

Approximate Computing (AC) is a set of techniques that trade off computation accuracy so as to achieve better performance or energy consumption. AC leverages the fact that applications like real time streaming and multimedia processing can tolerate compromises on image or video quality [35]. For instance the HTTP Adaptive Streaming (HAS) protocol streams videos with varying quality to adapt to changes in network (bandwidth limitations and traffic) and minimize stalls [36]. Such quality approximations may lead to increased user experience and savings in network bandwidth. Applications of AC are not limited to video streaming, but can be seen in Deep Learning, wireless communication and control systems.

We used approximate computing techniques to trade off video quality to adapt with the latency variations experienced in the wireless channel due to varying interference levels. We identified a set of video quality modification techniques that reduces the video frame size and hence result in decreased frame transmission latency through wireless channel. In the following part of this Section we describe prior research work that exploited approximate computing in various domains.

In [37], Mittal provides a survey of approximate computing techniques. Strategies for approximation at the code level such as loop perforation, and at the architecture level such as reduced precision operations are discussed. Regarding applications of approximate computing to Deep Learning, Chen et al. [38] use approximate computing to accelerate network training, while Ibrahim et al. [39] explore the use of approximate computing to realize Deep Learning networks on resource constrained embedded platforms. Unlike our work, in these works approximate computing is targeted towards reducing the computational load.

In [40], Betzel et al. introduce the concept of approximate communication to reduce the communication between processing elements in a high performance computing system. They evaluate compression, reduced synchronization, and value prediction as potential approximate communication techniques. In contrast to Betzel et al. we target latency variations due to interference in wireless communication channels, and investigate the impact on application accuracy.

2.3 Computer Vision Applications

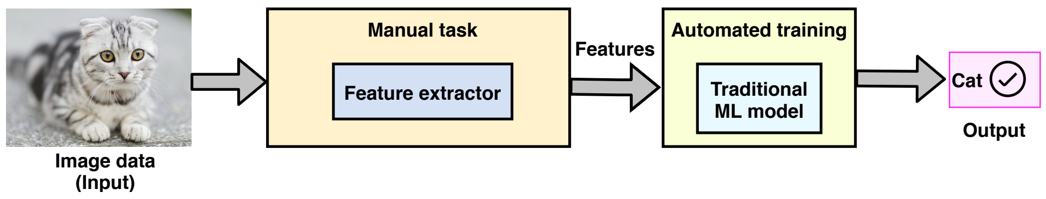
Computer vision aims to build autonomous systems by gaining high-level understanding from observed images or videos. Computer vision systems translate visual information obtained from images or videos into insights to make decisions and predictions. Computer vision is distinct from image processing in the way that, the latter is mechanical transformations used to make alterations to image properties such as resolution, brightness, contrast and so on. Whereas computer vision processes visual data and creates intuitive understanding to identify, classify or categorize the information in the visual data.

Prior to 2012, traditional computer vision techniques used a top-down approach to detect and/or classify information in the visual data [41]. Entities or objects in the visual data are composed of different features. For instance a cat in an image is a combination of several features such as head, ears, tail and four legs. Training a computer vision system to understand these features requires the system to have pixel level understanding of the visual data. The system analyzes images to obtain insights on minute differences on pixel density, color saturation, levels of darkness and brightness. Certain arrangement of pixels with certain level of brightness would indicate a cat's ear for instance.

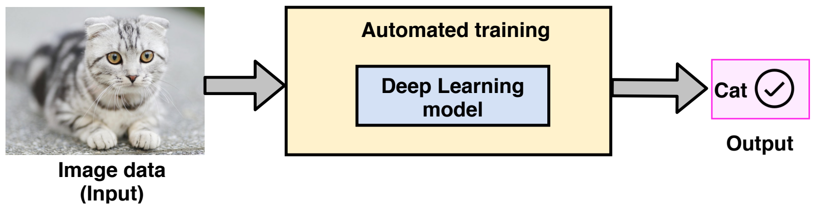
Such early systems were inflexible and time consuming to build because they involved manual effort to codify features (required for detection) for each object for each vision application. These systems used machine learning (ML) models and developers trained these models using the features they believed to be relevant to the objects. Developers had to explicitly tell the rules to the system that cats are made of four legs, two ears and a tail. Further minor changes in image clarity, object orientation and rotation would make these systems fail to perform detection or classification.

However recent advances in Deep Learning have changed the way computer vision systems built traditionally. Deep learning techniques use a bottom-up approach and are able to train a machine using massive datasets and numerous training cycles. For instance during the training process the deep learning algorithms automatically extracts relevant features of a cat. After the training phase a Deep Learning model is produced and can be used to detect and predict accurate information about previously unseen visual data. Figure 2.2a and 2.2b show the distinction between traditional computer vision based system and Deep Learning based system workflows for a simple object detection application.

Deep Learning in computer vision utilize large scale vision datasets such as ImageNet [42], CIFAR10 [43] and Microsoft COCO [19]. These datasets provide visual



(a)



(b)

Figure 2.2: (a) Traditional computer vision workflow: Involves manual task of telling the system what are the features of the objects need to be detected and training ML model using these features. (b) Deep Learning computer vision workflow: Deep Learning model automatically extracts the features and learns using these features.

data containing wide variety of objects. In this dissertation we have used three publicly available vision datasets namely JAAD [44], DukeMTMC [45] and Microsoft COCO 2017. Among these JAAD dataset contains over 300 video clips of pedestrians recorded at traffic intersections and public spaces using high resolution cameras. This dataset can be used to train and build Deep Learning models that perform computer vision tasks such as pedestrian detection and pedestrian path prediction. The DukeMTMC dataset consists of video clips of people recorded using 8 camera views in Duke University campus. Since DukeMTMC dataset contains videos from multiple camera perspectives, this dataset is useful to build models that perform multi-camera person re-identification and tracking. In contrast with these two datasets Microsoft COCO dataset contains images ($\approx 330K$) belonging to 80 different object categories. COCO dataset has been used to develop and validate variety of vision algorithms and models for object detection, segmentation and keypoint detection. All three datasets mentioned above provide labels for the objects in the images and videos. These labels are created by the dataset providers by manually annotating objects in the dataset. Such labels are termed as ground truth for the objects in the dataset.

When sufficiently large set of images are fed to a Deep Learning model it automatically learns the pixel-level details in the images such as varying color and contrast information, background and edges of objects. After gaining insights about features (head, tail and legs for cat) by learning pixel-level details the model returns its final outcome (for example cat).

In this dissertation we used two Deep Learning based computer vision applications - pedestrian detection and object detection. The pedestrian detection application uses OpenPose [13, 14, 15, 16] which is a multi-person system to detect human body, hand and facial key-points in images and videos. The pedestrian detection application draws bounding boxes around the key-point detected pedestrians outputted from OpenPose. The generated bounding boxes are compared against ground truth (provided by the dataset) of the objects. If bounding boxes for the detections and their ground truth match and their areas overlap above a certain threshold (IoU-Intersection over Union) then the detections are considered to be valid (true). For object detection application we used EfficientDet - an object detector model developed by Google Brain team [17]. EfficientDet outperforms state of the art object detector models in terms of model efficiency.

In this dissertation, we use the term inference accuracy to represent a vision application's Deep Learning model accuracy. Some commonly used metrics used to calculate inference accuracy are Average Precision (AP), mean Average Precision (mAP) and F1 Score. We define these metrics in their relevant Chapters in this dissertation.

2.4 Messaging Systems

A messaging system's function is to transfer data between multiple applications. In the messaging context, the application that sends data is called producer and application that receives data is called consumer. A common approach for notifying consumers about a new event is, the producer sends message containing the event, which is then pushed to consumers. A messaging system can be implemented by using a direct communication channel like a Unix pipe or TCP connection between producer and consumer. But Unix pipes and TCP connect exactly one sender with one recipient, whereas a messaging system need to have the ability to send messages to multiple consumers from a single producer and multiple producers.

A number of messaging systems such as ZeroMQ [46], StatsD [47] and Brubeck [48] use direct network communication (TCP and UDP) between producers and consumers. But these direct messaging systems require application code to be aware of the possibility of message loss. The faults they can tolerate are quite limited because they assume that producers and consumers are always online. If a consumer goes offline, messages are lost in the communication channel because they were sent when consumer was unreachable at that point.

An alternative to this problem is to use message brokers. A message broker runs as a server and producers and consumers connect to it as clients. Producers write messages to the broker and consumers can retrieve these messages by reading them from the brokers. By centralizing the data in the broker, messaging systems can more easily tolerate clients that frequently disconnect and crash. In this case the durability aspect of messages are handled by broker instead of applications.

Using a message broker has several advantages compared to direct messaging systems,

- It can act as a buffer if the consumer is unavailable or overloaded, and thus improve system reliability.
- It can automatically redeliver messages to a consumer that has crashed, and thus prevent messages from being lost.
- It avoids the producer needing to know the IP address and port number of the consumer.
- It allows one message to be sent to several consumers.

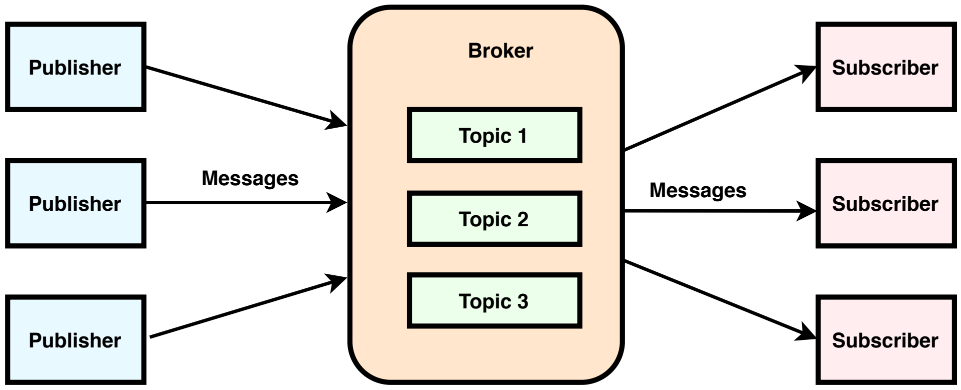


Figure 2.3: Architecture of a publish-subscribe system. Multiple publishers publish messages to a broker and several subscribers consume messages from the broker. Publishers categorize and publish messages to different topics in the broker.

- It logically decouples the producer from the consumer.

A publish-subscribe (pub-sub) messaging is a type of messaging pattern that is characterized by the producer (publisher) of a piece of data (message) not specifically directing it to a consumer. Instead, the publisher classifies the message based on a policy; the consumer (subscriber) subscribes to receive certain classes of messages. To facilitate this, pub-pub systems have a message broker, a central point where messages are published [49]. Publishers categorize and publish messages to different topics in the broker (see Figure 2.3). Further, pub-sub messaging ensures that each subscriber receives messages on a topic in the exact order in which they were received by the messaging system.

In this dissertation we present the design of a pub-sub messaging system - Mez, for latency sensitive multi-camera computer vision applications distributed at the IoT Edge. Mez provides vision applications the ability to specify their latency and accuracy demands and achieves these demands in presence of dynamic network conditions at the Edge. We now review state of the art messaging systems such as RabbitMQ, Kafka and NATS and indicate their major distinctions from Mez.

RabbitMQ [12] is an open source messaging system that supports the Advanced Message Queuing Protocol (AMQP), Streaming Text Oriented Messaging Protocol

(STOMP), Message Queuing Telemetry Transport (MQTT), and other protocols. It supports multiple messaging styles including pub-sub, request-reply, and point-to-point communication models. RabbitMQ's design assumes a smart broker, dumb consumer model, with the broker consistently delivering messages. Mez, in contrast, supports a dumb broker, smart consumer model, and is designed specifically for machine vision applications at the Edge.

Kafka proposed by Kreps et al. in [10] is used for collecting and delivering high volumes of data with high throughput. It combines the benefits of traditional log aggregators and messaging systems. Kafka is a pub-sub system in which multiple producers and consumers can publish and retrieve messages at the same time, and store streams of data in distributed, fault tolerant clusters using multiple brokers and partitions. Similar to Mez, Kafka supports a dumb broker, smart consumer model. However, Kafka is focused on delivering high throughput, and not necessarily on latency of individual messages.

NATS [11] messaging system is a recent project that is focused on providing low latency to cloud native applications. Similar to Mez, NATS supports a pub-sub system, and a dumb broker, smart consumer model. However, unlike Mez, NATS is a general purpose messaging system, and does not provide latency guarantees.

2.5 Control Systems

A control system is a mechanism that alters the future behaviour or state of a system. In order to consider a system as control system the behaviour or the outcome must tend towards a state that is desired. A control system has two basic components - the system to be controlled (plant) and the input that acts on the plant. For a given input the plant responds over time to produce a system output. There are mainly two types of control systems - open loop and closed loop. In an open loop control system, the input does not depend on the system output. The drawback of an open loop control system is that the input to the system has no way to compensate the variations in the system. To account for these changes, system input must varied with respect to the output. This type of control system is called a closed loop (feedback) control system. In closed loop control the output of the system is measured using a sensor and the output is compared against a reference signal (desired/commanded state). The error between measured and desired values is fed to a controller, where the error is converted into a system input value (see Figure 2.4). The advantage of feedback control system is that it is capable of reacting to changes to the plant automatically, by constantly driving the error term to zero.

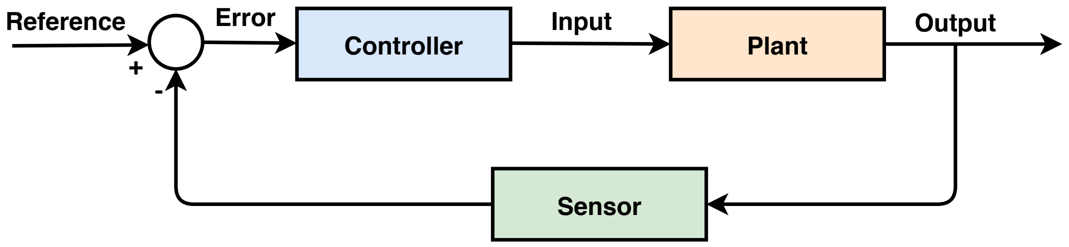


Figure 2.4: Block diagram of a feedback control system. Plant output is measured using a sensor and is compared against a reference signal. The error between measured output and reference signal is fed to a controller and is converted to plant input.

The controller in a control system can be designed in several ways. In the context of this dissertation, we describe a PI (Proportional-Integral) type controller. For a PI controller the controller output can be defined using Equation 2.1.

$$u(t) = K_p e(t) + K_i \int e(t) d(t) \tag{2.1}$$

where $u(t)$ is the controller output, K_p is proportional gain, K_i is integral gain, and $e(t)$ is controller error. The two tuning parameters for a PI controller are the proportional gain and integral gain. Controller error is the difference between measured and desired output. $K_p e(t)$ and $K_i \int e(t)d(t)$ are termed as the proportional (P) and integral (I) terms in the PI controller. The proportional and integral terms influence the controller to produce output close to the desired output. Term P is proportional to the controller error $e(t)$. Term I is generated by integrating the past values of controller errors over time. The balancing of both these terms is achieved by tuning the controller gains K_p and K_i. The tuning constants' values to achieve an optimal control depend on behaviour of the measuring sensor as well as plant output. To obtain optimal values for the constants, they have to be tuned by setting reference signal at different levels and observing system responses.

2.6 Machine Learning

Machine Learning (ML) is a set of tools for making inferences and predictions from data. Predictions can be defined as the outcomes of any future events. Inference refers to drawing insights from events and their behaviours. Machine Learning is a powerful technique because it gives computers the ability to learn without being explicitly programmed to do so. Essentially ML learns patterns from existing (training) data and applies it to new (testing) data. For machine learning to be successful it needs high-quality data.

A machine learning model can be defined as a statistical representation of a real-world process. A process is modeled using data and we can enter new inputs into a model to get an outcome. When a model is being built and learned from training data, we call this 'training a model'. Samples in training and testing data are called observations. Observations are comprised of features and targets. Features are different types of information that help the model to predict the targets. Targets are outcomes (quantities/categories) that take values depending on the features.

There are mainly three categories for machine learning - reinforcement learning, supervised learning and unsupervised learning. Reinforcement learning is used for deciding sequential actions. In supervised learning the training data is labeled, meaning the values of the targets are known. In contrast with this unsupervised learning has only features, but no labeled targets. Machine learning workflow primarily consists of 4 steps - extracting features from raw data and creating the input dataset for the model, split the dataset into two for training and testing, train the ML model using train dataset and evaluate and tune the model till the model performance is good enough (see Figure 2.5).

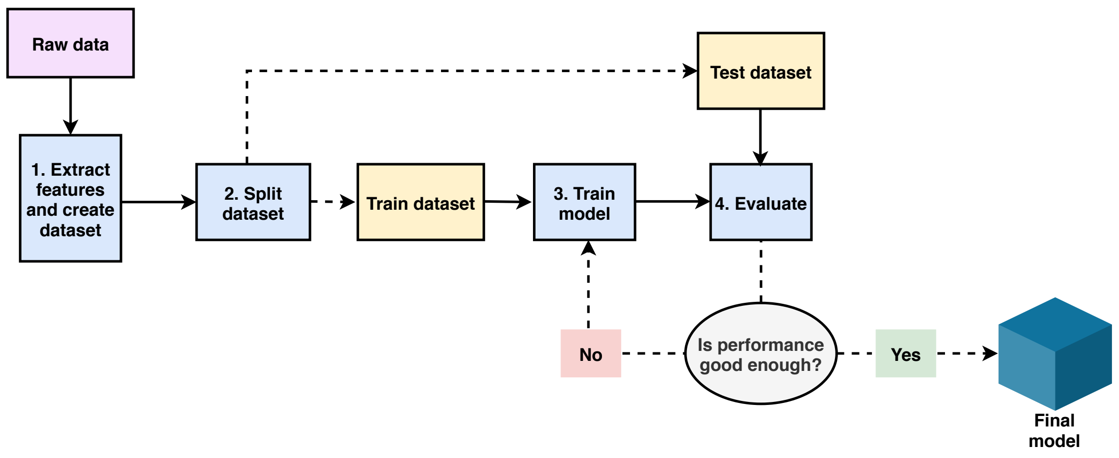


Figure 2.5: Basic machine learning workflow consists of 4 steps: extract features from raw data and create dataset, split the dataset into train and test dataset, train the ML model using train dataset, evaluate the model and if the performance is good enough create the final model.

Two categories of supervised learning are classification and regression. Classification models assign a category to an observation. In classification a discrete variable is predicted, meaning the target variable can take only a few different values. In contrast with this, regression models predict a continuous variable (a variable that can take any value). Common metrics used to evaluate regression model performance are R squared and RMSE (Root Mean Squared Error). In a dataset, if the target variable follows a linear relationship with the features, this dataset can be modeled using a linear regression model. Whereas if target-feature dependence is more complex, then non-linear models like polynomial regression, Support vector machine [50], Random forest regression [51] and Decision tree regression [52] can be used to model the data.

Boosting is one the techniques in ML that can be used to solve complex data driven real world problems. Boosting is defined as an ensemble learning technique that uses a set of ML algorithms to combine weak learners to form strong learners in order to increase the performance of a ML model. Ensemble learning is a method that is used to enhance the performance of ML model by combining several learners. When compared to a single model, this type of learning builds models with improved efficiency and accuracy [53]. In boosting weak learners are sequentially produced during the training phase. The performance of the model is improved by assigning a higher weightage to the previous incorrectly predicted samples. This process is repeated until all the mispredicted samples are correctly predicted.

The weak learners in boosting are generated by applying fundamental machine learning algorithms on different distributions of the dataset. In a boosting algorithm the weak (base) learners are decision trees by default. Decision tree is a graphical representation of all the possible solutions to a decision based on certain conditions. The base learners generate weak rules for each iteration. After multiple iterations the weak learners are combined and form a strong learner that predicts a more accurate outcome. Three types of boosting algorithms commonly used are adaptive boosting [54], gradient boosting [55] and XGBoost [56]. Among these, though adaptive boosting and gradient boosting can be used in both classification and regression problems, adaptive boosting is more commonly used to solve classification problems. In gradient boosting base learners are generated sequentially in such a way that the present base learner is always more effective than previous one. Thus in each step gradient boosting algorithms try to minimize the prediction error of the previous learner.

In this dissertation, we explored a recently proposed machine learning model called CatBoostRegressor [57] from open source gradient boosting library CatBoost [58]. This library is particularly useful for datasets that contain categorical features (tuning knobs in our case, see Section 6.1). Unlike numerical features categorical features represent a fixed set of discrete values with no mathematical correlation. CatBoost library is implemented using gradient boosted symmetric decision trees which help to reduce the prediction time for the machine learning model. CatBoost library outperforms state of the art gradient boosting libraries such as XGBoost [56] and

LightGBM [59] in terms of model quality and training speed.

2.7 Summary

In this Chapter we provided a brief overview on Edge computing, approximate computing, computer vision applications, messaging systems, control systems and machine learning. We also presented previously reported work on Edge computing, application of approximate computing and distributed messaging systems.

CHAPTER 3: NETWORK LATENCY CONTROLLER

In this Chapter we describe the IoT Edge test bed used to construct the Edge vision system shown in Figure 1.1. The test bed consists of low power embedded boards (IoT camera nodes) equipped with cameras communicating to a workstation (Edge server) equipped with GPU over Wi-Fi (802.11ac). We also present experimental study of impact on latency of video frames transferred over WiFi from IoT camera nodes to the Edge server (referred to henceforth as network latency for brevity) by two factors - peer IoT camera nodes' transmission and video frame size. We postulate that the network latency of video frames at the Edge substantially increases as the number of IoT camera nodes scale due to interference from peer camera nodes. We then conclude that for fixed camera node locations, tuning the video frame size is a potential means to modulate the network latency experienced by the video frames.

We use this observation to exploit the trade off between network latency and quality (approximate computing) of video frames. Approximate computing is based on the idea that in some applications, selective inaccuracies in computation can be tolerated to achieve gains in efficiency [37]. Machine vision applications can potentially make use of approximate computing since they can tolerate compromises on object/event detection accuracy resulting from selective loss of video frame quality.

In this Chapter we explore potential modifications that can be done on video frames through which we can modify the quality and thus vary the frame sizes. We call these modifications as video frame quality tuning knobs. We present 5 such tuning knobs in this Chapter - resolution, colorspace modifications, blurring, artifact removal and frame differencing. We then study the impact of these tuning knobs on inference accuracy of a pedestrian detection machine vision application based on OpenPose

[13, 14, 15, 16] using two publicly available datasets (JAAD [44] and DukeMTMC [45]).

We finally study the network latency experienced by the reduced size video frames and conclude that lower quality video frames (with smaller size) can be transferred with lower network latency, if they provide acceptable accuracy. We use this observation to design an approximate computing based latency controller that dynamically uses multiple video frame frame quality knobs to simultaneously maintain application specified network latency, and inference accuracy in the presence of interference in the Wi-Fi communication channel. While inputs to the latency controller are application specified network latency and inference accuracy bounds, the output is video frames that satisfy these bounds.

The organization of this Chapter is as follows:

1. Section 3.1 describes the test bed to characterize network latency at the Edge, characterization of the impact of peer IoT camera nodes on network latency when scene dynamics and frame rate of the video frames are varied and study of impact of video frame size on network latency.

2. Section 3.2 describes the video frame quality tuning knobs (resolution, colorspace modifications, blurring, artifact removal and frame differencing), characterization of the impact of tuning knobs on inference accuracy for perdestrian detection machine vision application and the study of impact of video frame quality on network latency.

3. Section 3.3 presents the design of building blocks of the latency controller and the latency control algorithm that maintains the application specified network latency in presence of interference, by automatically tuning the video frame quality knobs.

4. Section 3.4 presents the evaluation of the network latency controller for JAAD

and DukeMTMC complex scene dynamics video frames.

5. Section 3.5 summarizes this Chapter.

3.1 Characterization of Wi-Fi Latency at the Edge

In this Section we describe the IoT Edge test bed used to construct the Edge vision system shown in Figure 1.1. We also present the study of impact on latency of video frames transferred over Wi-Fi from IoT camera nodes to the Edge server (referred to as network latency) due to multiple factors - (1) interference by peer IoT camera nodes, (2) video scene dynamics, (3) video frame rate, and (4) video frame size.

3.1.1 IoT Edge Test Bed

We set up an Edge testbed similar to the IoT Edge machine vision system shown in Figure 1.1. Our Edge test bed consists of five IoT camera nodes equipped with 8-core ARMv8.2 based embedded Nvidia Jetson AGX Xavier [60] boards, and an Edge server. A workstation equipped with an Nvidia Titan V GPU [61] serves as the Edge server. The embedded boards and the workstation run Linux. The wireless link consists of a NETGEAR Nighthawk XR700 [62] access point that uses 802.11ac (5 GHz) Wi-Fi standard. The Edge server is connected to the access point through Ethernet, while the IoT camera nodes connect to the access point through the 802.11ac Wi-Fi link. The IoT camera nodes are placed at 6m from the access point.

We use two publicly available video datasets - JAAD [44] and DukeMTMC [45] for latency characterization. The JAAD dataset consists of videos of pedestrian movement in public spaces captured under various camera types and qualities in different weather/lighting conditions. The DukeMTMC data set consists of 1080p videos recorded at 60 fps from 8 static cameras deployed on the Duke University campus. To perform the Edge latency measurements, we chose video clips with three different scene dynamics - simple, medium, and complex, from both JAAD and DukeMTMC datasets.

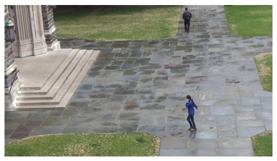
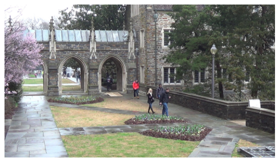
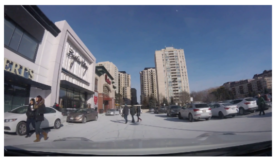
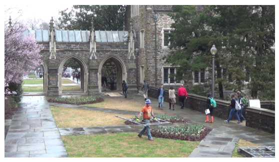
Dataset / SD	JAAD	DukeMTMC
S		
M		
C		

Figure 3.1: Sample images from JAAD and DukeMTMC dataset with simple (S), medium (M) and complex (C) scene dynamics (SD) showing pedestrians at public spaces such as traffic intersections, parking lots, and public buildings.

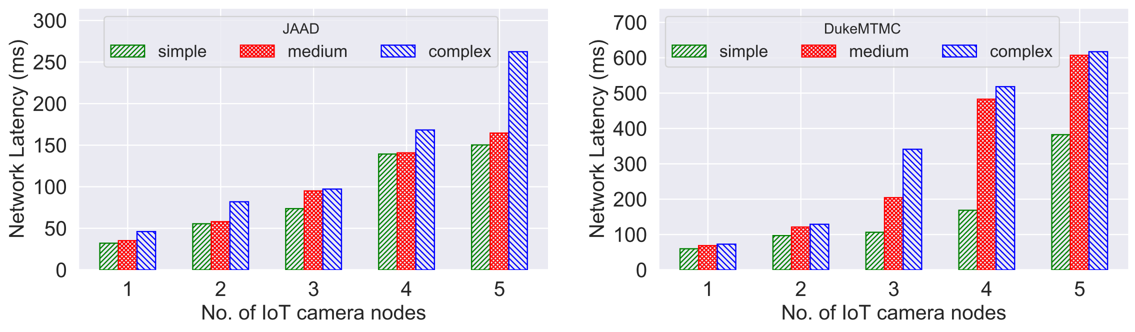
In order to cluster the video frames in the two data sets as simple, medium and complex, a k-means clustering [63, 64] approach was implemented. For DukeMTMC, all frames from two of the cameras (cameras 5 and 6) were k-means clustered using the mean and standard deviation of the bounding box areas in a scene. A frame sequence of 100 frames was then randomly selected from each cluster for network latency measurements. A similar approach was followed for the JAAD dataset. Figure 3.1 shows a representational sample of images from JAAD and DukeMTMC datasets.

A Golang gRPC [65] based client and multi-threaded server were deployed at the IoT camera node and the Edge server respectively to facilitate video frame transfer, and perform network latency measurements. The wireless network latency of video frame transfer is measured by sending timestamped video frames from IoT camera node to the Edge server. The latency is calculated as time difference $t_{Received} - t_{Send}$. The IoT camera nodes are time synchronized to the Edge server before starting the network latency measurements using the PTP network level time synchronization protocol capable of microsecond accuracy [66].

3.1.2 Impact of Peer IoT Nodes on Network Latency

In the measurements described below, we measure the network latency experienced by IoT camera node 1 (see Figure 1.1) due to the 4 peer IoT camera nodes. All latency measurements are at the 95th percentile with video frames transmitted at 5 fps.

Figure 3.2 shows the per frame network latency measured at the test camera node as the number of IoT camera nodes transmitting video frames (with simple, medium and complex scene dynamics) is increased from 1 to 5. Table 3.1 summarizes the latency measurements. ONE_{Lat} is the per frame network latency to the Edge server when only the test node is active. $FIVE_{Lat}$ is the per frame network latency when the node under test and the 4 peer camera nodes transmit video frames to the Edge server. We note that for video frames with complex scene dynamics, the ratio $FIVE_{Lat}/ONE_{Lat}$ is 5.6x for the JAAD dataset, and 8.4x for the DukeMTMC dataset.



(a) (b)

Figure 3.2: Characterization of the impact of peer interference on the video frame transfer latency for frames with different scene dynamics from (a) JAAD and (b) DukeMTMC datasets. For complex scene dynamics 5.6x and 8.4x increase in latency is observed for the JAAD and DukeMTMC datasets respectively.

We also investigate the impact of peer node interference at higher video frame rates and with increasing distance of IoT camera nodes from the Edge server. Table 3.2 compares the network latencies between 5 and 15 fps for complex scene dynamics video frames from the DukeMTMC dataset at both 6m and 12m. We note that the

$FIVE_{Lat}$ at 15 fps is 1.02x higher for DukeMTMC compared to 5 fps, and at 12m is 1.06x higher compared to 6m.

Table 3.1: Summary of impact of video scene dynamics on network latency for JAAD and DukeMTMC workloads with simple (S), medium (M) and complex (C) scene dynamics (SD) video frames. All latency measurements are at the 95th percentile with video frames transmitted at 5 fps. ONE_{Lat} is the per frame network latency to the Edge server when only the test node is active. $FIVE_{Lat}$ is the per frame network latency when the node under test and the 4 peer camera nodes transmit video frames to the Edge server.

Dataset	JAAD			DukeMTMC		
SD	S	M	C	S	M	C
$Size_{med}$ (KB)	610	760	970	1390	1670	1740
ONE_{Lat} (ms)	32.09	35.16	46.09	59.71	68.73	72.72
$FIVE_{Lat}$ (ms)	150.28	164.56	262.43	382.47	606.98	617.16
$FIVE_{Lat}/ONE_{Lat}$	4.6x	4.6x	5.6x	6.4x	8.8x	8.4x

Table 3.2: Summary of network latency vs. frame rates (5 and 15 fps) and distance from Edge server (6m and 12m) for DukeMTMC complex scene dynamic video frames.

Num. nodes	Network latency (ms)		
	5fps (at 6m)	15fps (at 6m)	5fps (at 12m)
1	72.72	80.60	96.35
2	128.97	409.82	162.15
3	341.18	438.01	390.75
4	518.31	585.58	526.95
5	617.16	631.76	657.88

The measurement results indicate that in an IoT machine vision application with multiple cameras transmitting video frames to the Edge server, a significant rise in network latency is observed at each IoT node as the number of peer nodes scale. Additionally, factors affecting latency include scene dynamics; frame rate, and distance of IoT camera nodes from the Edge server are less significant. In a real-world deployment, additional external interference effects from unrelated transmitters in the neighborhood of the deployment worsen the latency. Moreover, the network latency is dynamic due to scene changes (simple to complex), and the intermittent nature of external interference.

3.1.3 Impact of Video Frame Size on Network Latency

Size of video frames varies depending on the information content present in them. In this Section we explore the latency experienced by video frames with different sizes in the Wi-Fi channel using the IoT Edge testbed (shown in Figure 1.1).

We characterized the network latency of video frames when they are transmitted from a test camera node (Node 1 in Figure 1.1) to the Edge server while keeping all other camera nodes (Nodes 2 to 5) inactive. This setup emulates an operational scenario where cameras produce variable size video frames depending on the scene dynamics in the area of observation.

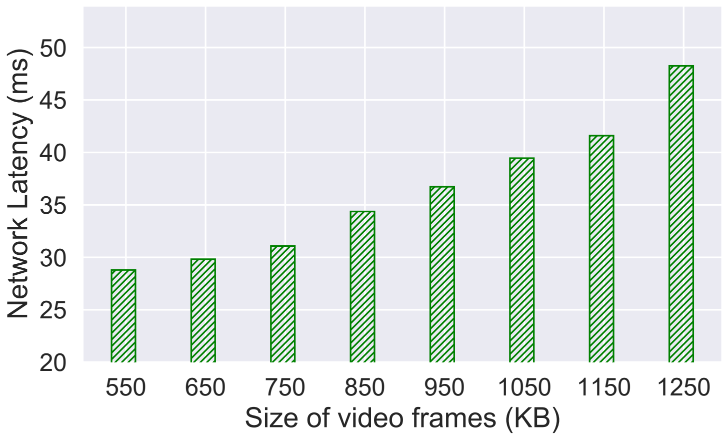


Figure 3.3: Characterization of the impact of video frame size on video frame transfer network latency. Network latency shows an approximately linear variation with video frame size.

Figure 3.3 shows the variation in network latency at different video frame sizes. The video frames of different sizes are chosen from JAAD and DukeMTMC datasets and each measurement is taken as the average of 10 measurements. From this measurements, we note that the network latency shows an approximately linear variation with video frame size. Video frames with reduced size can be potentially transmitted with reduced network latency. This results suggests that for camera nodes at fixed locations, the latency can be tuned by varying the size of the video frame. In the next Section we explore various techniques to tune the video frame size.

3.2 Approximate Computing for Latency Control

Approximate computing (AC) exploits the gap between the level of accuracy required by the applications/users and that provided by the computing system, for achieving diverse optimizations [37]. AC leverages the fact that several important applications, like machine learning and multimedia processing, do not necessarily need to produce precise results to be useful. In these applications, we can drop some video frames or lower the frame resolution, provided that, the machine vision applications' inference accuracy does not suffer substantially.

As seen in our experimental evaluation of latency in Section 3.1.2, channel interference from other camera nodes can cause network latency to increase. We also note that for fixed camera node locations, tuning the video frame size is a potential means to control latency. However, reducing the information content in the video frames could make them unusable for object/event detection/prediction vision applications. In this Section we explore potential modifications that can be done on video frames through which we can modify the information content and thus vary the frame sizes. We call these modifications as tuning knobs for video frames. We present five such tuning knobs - resolution, colorspace modifications, blurring, artifact removal, and frame differencing techniques. We then study the impact of these tuning knobs on network latency and inference accuracy of a pedestrian detection machine vision application in Sections 3.2.2 and 3.2.3.

3.2.1 Video Frame Quality Tuning Knobs

We use the open source computer vision library OpenCV [67] to explore different lossy image transformation techniques that can be applied to video frames to modify the frame size. We choose 5 such transformation techniques (which we call tuning knobs [68]). These are described below:

1. **Knob1 - Resolution:** Video frame size can be reduced by decreasing its resolution while keeping the aspect ratio constant. The cv2.resize() function from OpenCV downscales an image to the specified resolution. We choose the resolutions 1312x736, 960x528, 640x352, and 480x256 as possible knob settings. Modifying resolution can reduce the video frame size by as much as 84%.

2. **Knob2 - Colorspace modifications:** Video frames can be converted from one colorspace to another (using cv2.cvtColor() function from OpenCV) resulting in total size reduction. There are more than 150 color-space conversion methods available in OpenCV. We choose BGR↔Gray, BGR↔HSV, BGR↔LAB and BGR↔LUV colorspace modifications as possible knob settings. Our choice of color space modifications can reduce the video frame size by as much as 62%.

3. **Knob3 - Blurring:** Video frames can be blurred by passing them through various low pass filters. The cv2.blur() method from OpenCV blurs an image using normalized box filter. We choose filter kernel sizes of (5,5), (8,8), (10,10) and (15,15) as possible knob settings. Blurring the video frames can reduce the video frame size by as much as 46%.

4. **Knob4 - Artifact removal:** For cameras mounted at fixed positions, the background of recorded video stream is largely static over consecutive frames in the video. Thus video frame size can be reduced by removing the static background with stationary artifacts in it. First setting of this knob uses motion detection to detect and preserve moving objects in video frames, as well as to perform background subtraction to remove all the stationary objects from the video frames. In the second setting of this knob, we detect moving objects in the video frames and retain only their contours. Knob4 uses a combination of the OpenCV functions cv2.absdiff(), cv2.threshold(), cv2.dilate(), and cv2.findContours() to perform the above video frame modifications. Removing

artifact information can reduce the video frame size by as much as 98%.

5. **Knob5 - Frame differencing:** We applied frame differencing (using cv2.absdiff() function from OpenCV) on pixel values between pairs of consecutive video frames to selectively drop frames. We hypothesize that dropping video frames with similar content (within a threshold) will not adversely affect the machine vision task. We choose 5 knob values ranging from 0 to 0.72, where 0 represents pixel wise identical frames, and 1 represents completely dissimilar frames. For a stream of 100 simple dynamics images from the JAAD data set, this knob reduces the median image size by up to 40%.

In the next Section we study the impact of these tuning knobs on network latency.

3.2.2 Impact of Video Frame Quality on Network Latency

We investigate the impact on network latency when video frames with degraded quality are transferred from the IoT camera node to the Edge server. The degradation is caused due to discarding of information from the the video frame, resulting in a lower video frame size that can be potentially transmitted at reduced network latency. However, the lower frame size could adversely impact the accuracy of the machine vision application as well. The impact on the accuracy is application specific and will be evaluated in the context of a specific application in Section 3.2.3.

The application of combinations of the 5 tuning knobs identified above result in different sizes of video frames, all lower than the original. Figure 3.4 shows the resulting impact on the network latency (95th percentile) from the IoT camera node to the Edge server. The measurements were done by applying multiple tuning knob combinations (935 in all) to video frames drawn from the JAAD and DukeMTMC dataset. From Figure 3.4 we note that the Wi-Fi transmission latency shows an approximately linear variation with video frame size. A 4x reduction in video frame size could potentially yield a 4x reduction in wireless network latency. We also note

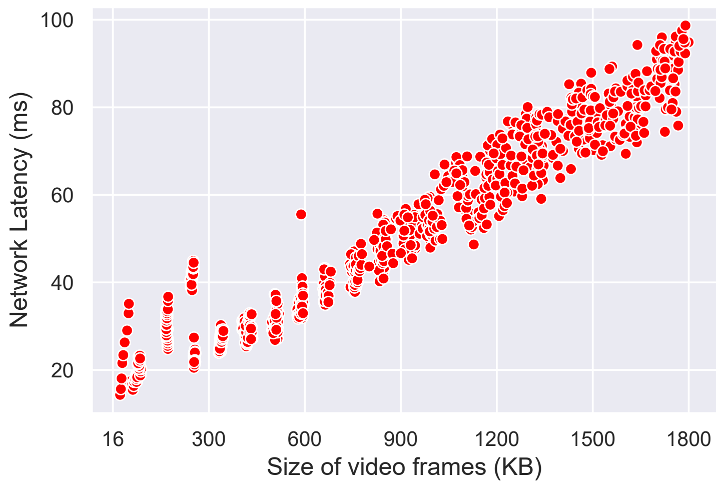


Figure 3.4: Network latency vs. video frame size. Video frame sizes are obtained by the application of different combinations of the 5 tuning knobs that modify the frame quality.

that multiple knob combinations map to the same video frame size (and hence network latency). However, these knob combinations could result in different application inference accuracy - which we characterize in the next section.

3.2.3 Impact of Video Frame Quality on Inference Accuracy

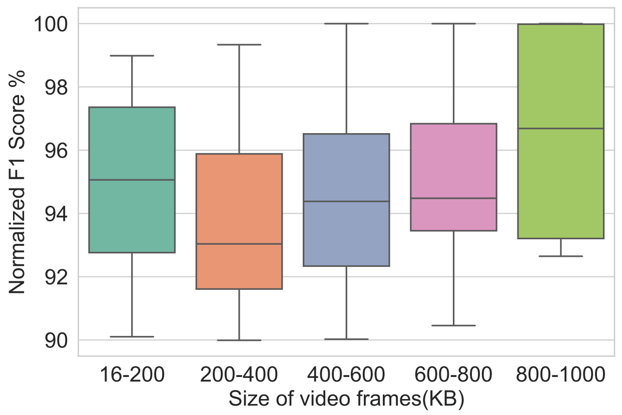
While we note the ability of tuning knobs to reduce network latency by reducing the size of the video frames, the question remains as to the impact of the lower sized video frames on the accuracy of the machine vision task. In general, the impact is dependent on the particular machine vision application. We evaluate the impact on pedestrian detection application accuracy using OpenPose with video frames drawn from the JAAD and DukeMTMC datasets.

The OpenPose project from CMU [13, 14, 15, 16] is an open source real-time multi person system to detect human body, hand and facial keypoints ((x,y) coordinates of different body parts) on individual images. We input the original and modified video frames from JAAD and DukeMTMC dataset with simple, medium and complex scene dynamics to OpenPose to generate the pose detected video frames and keypoint locations. From these keypoints, bounding boxes are created for each detection with the top-left and bottom-right most coordinates. A set of resulting bounding boxes is presented as the final output. In order to evaluate these detections, each ground truth bounding box for that frame (available for the two datasets) is matched exclusively to the outputted bounding box based on highest Intersection over Union (IoU) overlap. Positive matches with an IoU greater than a threshold are considered True Positives; result bounding boxes without ground truth matches are considered False Positives; and each unmatched ground truth box is considered a False Negative. These records are utilized for the F1 score calculation.

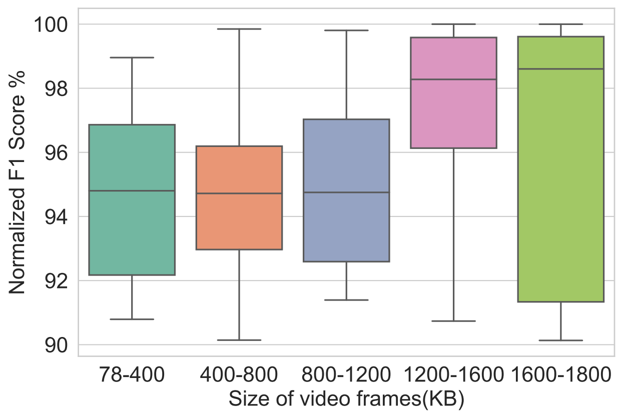
For pedestrian detection, we utilize the F1-score metric with an Intersection-over-Union (IoU) threshold of 0.5 as the application inference accuracy metric. Equation 3.1 defines the calculation for F1. Precision is $\frac{TP}{(TP+FP)}$ and Recall is $\frac{TP}{(TP+FN)}$, where TP, FP, and FN are the number of True Positives, False Positives, and False Negatives respectively.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3.1}$$

We evaluate the impact of the tuning knobs on the video frame size and pedestrian detection accuracy (F1) for JAAD and DukeMTMC datasets. To do this, we first calculate the F1 score for modified video frames (for all knob combinations) and normalize it with the baseline F1 score of unmodified video frames. Figure 3.5 shows the plot of the normalized F1 expressed as a percentage vs. video frame size for JAAD and DukeMTMC datasets. The video frame size buckets in Figure 3.5 corresponds to different combinations of the knob settings with different resulting accuracy. Note that higher F1 indicates higher accuracy. We have excluded knob combinations with resulting accuracy of less than 90%. This reduces the total knob combinations to 159 for JAAD and 140 for DukeMTMC. Further, due to the computationally intensive



(a)



(b)

Figure 3.5: Normalized F1 score expressed as a percentage for Openpose pedestrian detection application from (a) JAAD and (b) DukeMTMC datasets. Note that each video frame bucket corresponds to different combinations of the knob settings with different resulting accuracy.

nature of knob 4, we exclude knob 4 to maintain the video frame modification overheads to under 10 ms, bringing the total knob combinations to 70 for JAAD and 92 for DukeMTMC.

From the measurements in Sections 3.2.2 and Sections 3.2.3 we note the possibility of compensating for the increased network latency in the presence of channel interference through reducing the video frame quality just sufficiently, such that application accuracy demands are met (where feasible). This is an example of the paradigm of approximate computing - where despite computational approximations (video frame quality in our case) acceptable performance (application accuracy) can be obtained while gaining on another performance metric (network latency). Note that the limits of the trade-off needs to be determined and characterized by the application developer for a particular application of interest. We utilize the above observed trade-off between network latency and application accuracy in designing a network latency controller. Under dynamically varying network latency conditions, this latency controller automatically adjusts video frame quality such that the application specified network latency, and accuracy bounds are met.

3.3 Latency Control Algorithm

In this section we describe the algorithm that maintains the application specified video frame transmission latency from the camera to the Edge server in presence of interference, by automatically tuning the video frame quality knobs identified in Section 3.2.1. The control mechanism constructively reduces video frame size, to match the measured video frame transfer latency with the target latency specified by the application, while maintaining the inference accuracy request within limits throughout the operation.

The camera nodes shown in Figure 1.1 need to be able to provide video frames within the latency and inference accuracy levels requested by the vision applications executing on the Edge server. Since the dependence of application accuracy is complex (see Section 3.2.3), we have two options - (1) Use a sophisticated machine learning model to predict the accuracy and knob combinations for an input video frame size, or (2) Use a look up table that stores the video frame size and application accuracy for all knob combinations. We chose the lookup table approach since the total knob combinations of the 5 knobs results in 2500 values, a small number easily stored in memory. These can be initially characterized and quickly looked up using two hashtables - a primary hashtable with the video frame size as the key, and the candidate accuracies as the value, a secondary hashtable that uses inference accuracy as the key and knob settings as the values.

The control algorithm is outlined in the psuedo code shown in Listing 1.

Algorithm 1: Latency control algorithm

```
Result: Image quality knob setting
latencyTarget;
accuracyTarget;
errorThreshold;
nominalImageSize ← RegressionModel(latencyTarget);
latencyError ← latencySampled - latencyTarget;
while latencyError > errorThreshold do
    imageSize = nominalImageSize + K1*latencyError +
     K2*latencyErrorIntegral;
    accuracy ← BinarySearchTree.search(imageSize);
    knobSetting ← HashTable.lookup(accuracy);
    if accuracy > AccuracyTarget then
        return knobSetting;
    else
        return(No feasible solution);
    end
    latencyError ← latencySampled - latencyTarget;
end
```

The Edge latency controller running on the camera nodes periodically samples the video frame transfer latency (network latency) to verify if it is under the requested limit. The control is implemented in two steps - In Step 1, the error (error and integral of error for Proportional-Integral (PI) control) between the observed the the specified latency is used to determine the largest video frame size that can potentially satisfy latency requirements. K1 (proportional) and K2 (integral) in Algorithm 1 are the PI controller tuning parameters. The almost linear dependence of latency on video frame size (see Section 3.1.3) facilitates an efficient binary search for the nominal video frame size. In Step 2, the frame size is then used as a key in a look-up table to determine the associated application accuracy. A secondary look-up table uses the application accuracy obtained from the first lookup as key to determine the corresponding tuning knob settings. The lookup tables are implemented with Binary Search Trees and Hash tables to facilitate efficient queries. The video frames transmitted from the IoT camera node to the Edge server are modified based on the knob combinations using OpenCV libraries. The network latency is measured again at the next sampling interval, and if the error exceeds a preset threshold, Steps 1 and 2 are repeated.

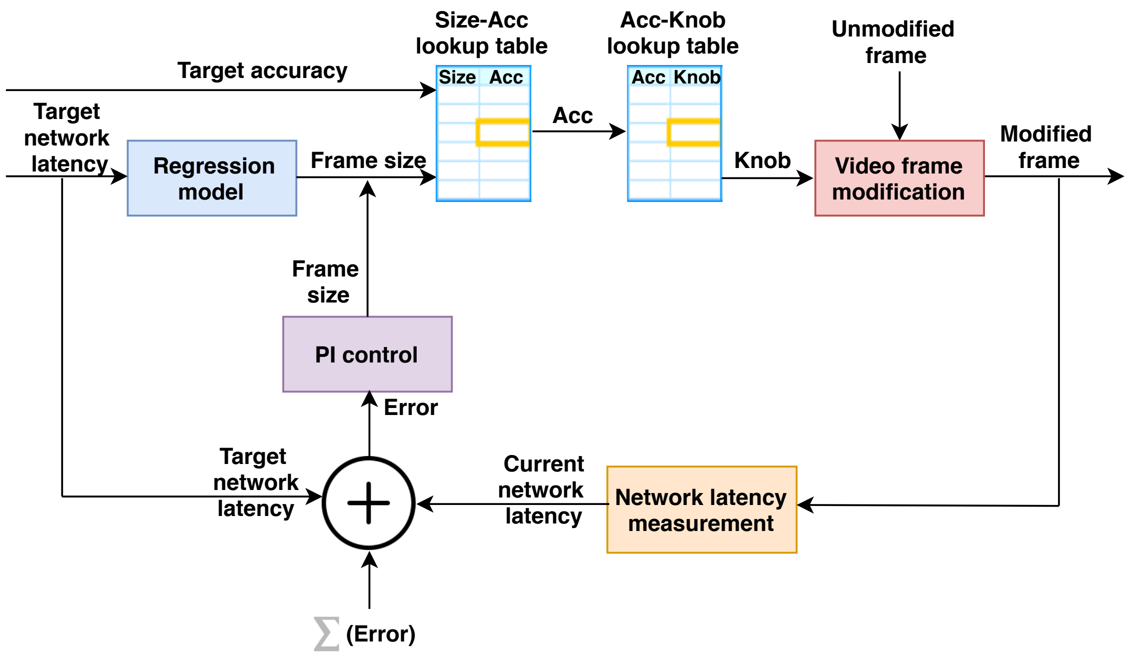


Figure 3.6: Block diagram of the network latency controller

If the application requested latency and accuracy are infeasible, the application is notified. At this point, the application has to decide whether to continue operation with relaxed requirements, or send the notification higher up the stack to the user.

Figure 3.6 shows the block diagram of the latency controller. The machine vision application informs the latency and inference accuracy demands (target network latency and target inference accuracy) to the latency controller. The regression model is constructed from the linear dependence of network latency on video frame size described in Section 3.1.3.

3.4 Evaluation

In this Section we present the evaluation of the proposed network latency controller. The IoT Edge test bed used in the evaluation is described in Section 3.1.1. It consists of one Edge server, and five IoT camera nodes connected to the Edge server through Wi-Fi (802.11ac). All latencies are measured between one of the IoT camera nodes, and the Edge server. The workload used is the pedestrian detection application with OpenPose described in Section 3.2.3 using video frames from the JAAD and DukeMTMC datasets described in Section 3.1.1. All latencies are measured at the 95th percentile with video frames streamed from the IoT camera node to the Edge server at 5 fps. The latency is calculated as time difference $t_{Received} - t_{Send}$. The camera nodes and the Edge server are synchronized using PTP synchronization protocol [66] before the start of the measurements.

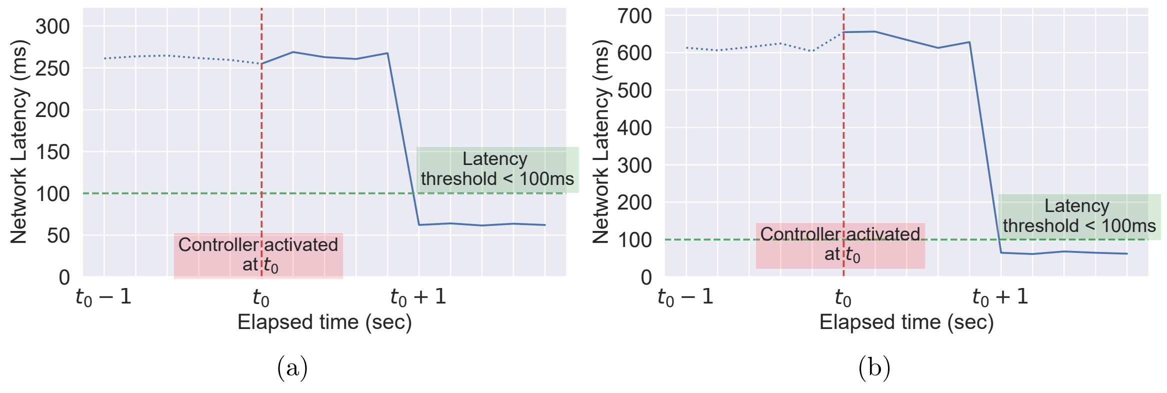


Figure 3.7: Latency controller step response for JAAD (a) and DukeMTMC (b) complex scene dynamics video frames. The measurement is taken with one test IoT node, and 4 peer IoT nodes.

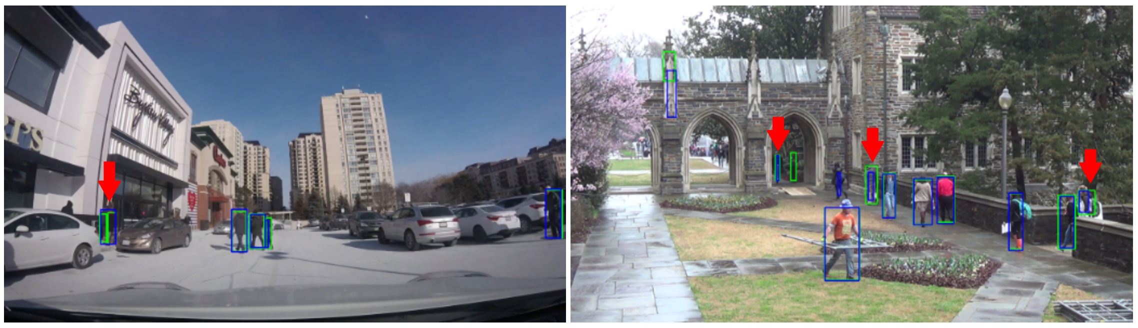
To evaluate the latency controller, the desired latency threshold is set under 100ms, and the desired application accuracy (normalized F1 score) is set above 95%. Figure 3.7a shows the step response of the controller for complex scene dynamics video frames from the JAAD dataset. With no latency control, the median latency is 260ms due to interference from the 4 peer camera nodes. With the controller enabled, the median latency is less than the latency threshold of 100ms. The controller is able achieve an application accuracy of above 96%.

Figure 3.7b shows the controller step response for the DukeMTMC dataset with complex scene dynamics. With no latency control, the median latency is 650ms for complex scene dynamics due to interference from the 4 peer camera nodes. With the controller enabled, the median latency is less than the latency threshold of 100ms. In all cases, the controller is able to achieve an application accuracy of above 95% with a settling time of less than one second. Table 3.3 summarizes the latency reduction and the resulting F1 score for JAAD and DukeMTMC datasets for all scene dynamics achieved using the proposed controller.

Table 3.3: Summary of median video frame size after modification by the latency controller, ($Size_{med}$), Normalized F1 Score expressed as a percentage, 95th percentile latency reduction with controller (Lat_{red}) for JAAD and DukeMTMC dataset for simple (S), medium (M) and complex (C) scene dynamics (SD).

Dataset	JAAD			DukeMTMC		
SD	S	M	C	S	M	C
$Size_{med}$ (KB)	124	173	228	293	172	371
F1 score (%)	99.1	98.3	96.7	98.9	96.7	95.8
Lat_{red}	6.8x	7.2x	4.1x	7.9x	9.5x	10.1x

Figure 3.8 gives a qualitative illustration of the effect of the inference accuracy loss with video frames from JAAD (Figure 3.8a) and DukeMTMC (Figure 3.8b) datasets with complex scene dynamics. Pedestrian detections for the unmodified video frames (green), and after video frame modification (blue) by the network latency controller are shown (accuracy loss of 3.3% for JAAD and 4.2% for DukeMTMC). For the JAAD video frame, a detection error occurs at the area indicated by the red arrow. A group of two individuals are detected correctly in the unmodified video frame, but detected as a single box in the modified video frame. For the DukeMTMC frame, the detection boxes in the modified video frame are thinner than those in the unmodified



(a) (b)

Figure 3.8: The impact of accuracy reduction on video frame modification by the latency controller for JAAD (a) and DukeMTMC (b). Green and blue bounding boxes indicate pedestrian detections on unmodified and modified video frames respectively. The video frames experience an accuracy loss of ≈3% for JAAD and ≈4% for DukeMTMC respectively. The red arrows show the resulting detection errors.

video frame. We note that for both datasets, apart from the aforementioned detection errors, the other detections in the modified frame are same as the unmodified frame.

3.5 Summary

In this Chapter, we demonstrated how network latency and inference accuracy specifications of machine vision applications at the Edge can be achieved despite the presence of significant latency variations due to interference in the wireless channel. The tuning knobs are derived from the approximate computing paradigm that a degraded video frame quality can be tolerated as long as application accuracy requirements are satisfied. We proposed an efficient two-step control algorithm that uses a proportional integral controller, and hashtable based lookups to dynamically determine the tuning knob settings based on latancies sampled during operation. Our control approach is scalable since each camera node runs its controller independently. Our experimental results on an Edge test bed with pedestrian detection machine vision application show that the proposed controller can correct for latency increase of upto 10.1x with an accuracy degradation of only 4.2%.

CHAPTER 4: MESSAGING SYSTEM ARCHITECTURE

Chapter 3 presented the design and evaluation of the network latency controller that adjusts video frame size using video frame quality tuning knobs to satisfy machine vision application specified network latency and inference accuracy requirements. The latency controller operates at the IoT camera nodes and the machine vision application is deployed at the Edge server in the Edge vision system (see Figure 1.1). In this Chapter we introduce an IoT Edge middleware [69] layer that provides a suitable abstraction for machine vision application developers to specify latency, and application inference accuracy bounds.

The middleware incorporates the network latency controller (described in Chapter 3) and then makes the best-effort to guarantee the specified latency and accuracy bounds by exploiting the latency-accuracy trade-offs inherent in machine vision applications (explained in Sections 3.2.3 and 3.2.2).

We call the IoT Edge middleware Mez [70]. Mez is a publish-subscribe (pub-sub) messaging system with storage capabilities. In Mez, publishers are cameras and subscribers are machine vision applications. Mez employs adaptive computing techniques and domain specific design decisions to adapt with the operating conditions present in the wireless channel.

In this Chapter, first we describe Mez API, data model and internal architecture. We then explain how the network latency controller (explained in Chapter 3) is incorporated in Mez. Later we provide insight into design of brokers in Mez, in-memory log storage and fault tolerance aspects.

4.1 API and Architecture

In this Section we present the Mez API that publishers and subscribers use to interact with Mez. We then provide an overview of the data model supported by Mez, and describe the architecture of Mez.

4.1.1 API

Mez has a simple API interface consisting of 5 API calls. As shown in Figure 4.1 - the APIs are Connect, Publish, GetCameraInfo, Subscribe, and Unsubscribe.

Connect*(url)* → *ID*
allows publishers and subscribers to connect with Mez and returns a unique ID

Publish*(videoStream)*
allows publishers to publish video stream with Mez

GetCameraInfo*()* → *list[cameraIDs]*
allows subscribers to get IDs of publishers connected to Mez

Subscribe*(applicationID, cameraID, tStart, tStop, latency, accuracy)* → *videoStream*
allows subscribers to receive video stream between two timestamps from Mez by specifying latency and accuracy bounds

Unsubscribe*(applicationID, cameraID)* → *status*
allows subscribers to stop receiving video stream from Mez and returns a success/failure status

Figure 4.1: A summary of the API provided by Mez.

Connect API is used by publishers (IoT camera nodes) and subscribers (machine vision applications) to connect to Mez. Publishers and subscribers are assigned a unique ID by Mez. Publish API allows publishers to push a stream of time stamped video frames to Mez. The GetCameraInfo API is used by subscribers to discover publishers. Subscribe API is used by subscribers to receive streaming video frames generated by a specific publisher. Additionally, the subscribers can specify begin and end times for the subscription, along with the desired latency and accuracy bounds for the video stream. Note that the end time could be in the future, in which case Mez delivers video frames as they become available. The Unsubscribe API is used by subscribers to terminate an ongoing subscription.

4.1.2 Data Model

Mez's data model is a simple key-value pair. The keys are the timestamps of a video frame, and the values are individual video frames. Video frames are stored in Mez in the same chronological order in which they are received from the publisher. Mez supports at-most-once delivery of video frames to the subscriber to limit the bandwidth consumption on the wireless channel. Any resend requests need to be done by the subscriber at the application level, since only the application can determine if a resend is needed considering task deadlines, and redundancy in the video frames.

4.1.3 System Architecture

The Mez system model consists of multiple IoT camera nodes, and an Edge server connected by a wireless network. The machine vision applications run on the Edge server, which has considerably more processing power than the IoT camera node.

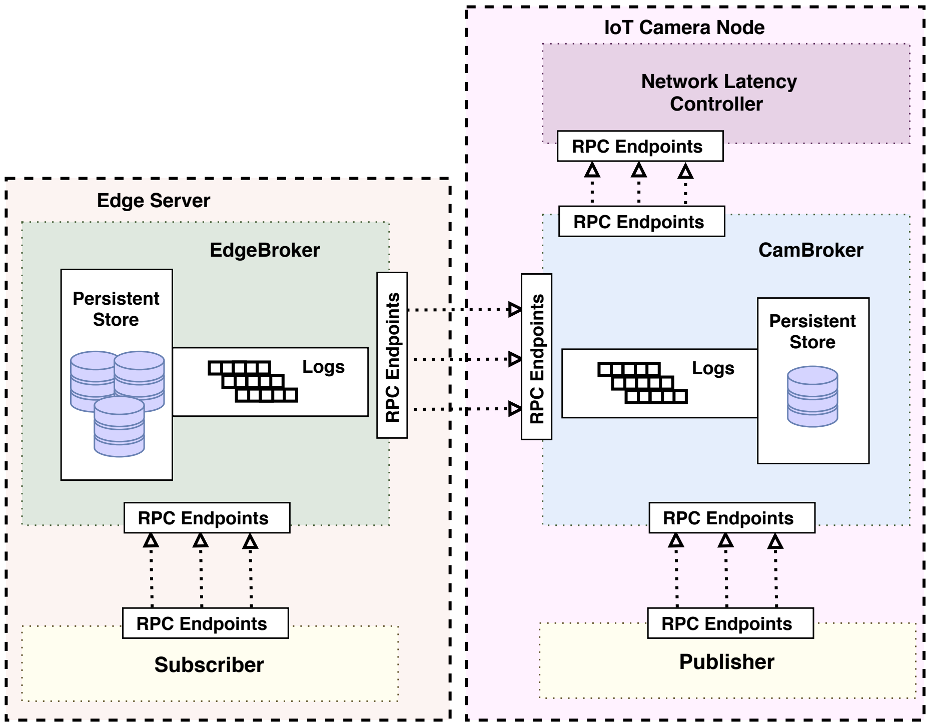


Figure 4.2: Detailed architecture of Mez

As shown in Figure 4.2, Mez consists of 3 components - a message broker, an in-memory log, and a network latency controller. The message broker implements the Mez API, the in-memory log is used to store video frames, and the network latency controller monitors wireless channel conditions, automatically adjusting the video frame quality to meet the application specified latency, and accuracy requirements.

A key architectural feature of Mez is the replication of the in-memory logs between the IoT camera nodes and the Edge server. The timestamped video frames generated by the publisher are initially stored in the in-memory log associated with the camera node. Maintaining the log at the IoT camera node allows buffering of video frames during conditions of intermittent connectivity with the Edge server. Upon a subscription request to the video stream by the application, video frames are transferred from the IoT camera node log to the Edge server log. The network controller resides on the IoT camera nodes, and modifies the video frames transmitted to the Edge server to satisfy the latency-accuracy requirements of the subscriber. The on-demand transfer of video frames reduces channel interference by limiting unneeded transmission in the wireless channel. Additionally, the reduced transmission also serves as a power saving opportunity at the IoT camera node. The in-memory logs are persisted on durable storage (SSD/disk) on both the IoT camera nodes, and the Edge server. However, to minimize storage latency, all requests are served from the in-memory log. The persistent storage is only used to reconstruct the in-memory log during reboot after node failure (see Section 4.2.4).

4.2 Messaging System Design

In this section, we present the detailed design of the different components of Mez - brokers, in-memory log, and the network latency controller.

4.2.1 Brokers

The brokers implement the Mez API, and interface with the log storage. Independent brokers are present on the Edge server and the IoT camera node. The broker on the Edge server (EdgeBroker) implements all the APIs shown in Figure 4.1 except Publish. Additionally, it implements two internal APIs, Register, and Unregister for IoT camera nodes to register/unregister with the Edge Server. The broker on the IoT camera node (CamBroker) implements all the APIs shown in Figure 4.1 except GetCameraInfo. The CamBroker also interfaces with the network latency controller on the IoT camera node. All APIs are implemented using gRPC [65] with TLS (Transport Layer Security) [71] for authentication and encryption of data in transit.

4.2.2 Network Latency Controller Integration

Figure 4.3 shows the block diagram of the latency controller incorporated in Mez. The CamBroker sends the latency and accuracy demands (target network latency and target accuracy) from the consumer application to the latency controller through an internal SetTarget API call.

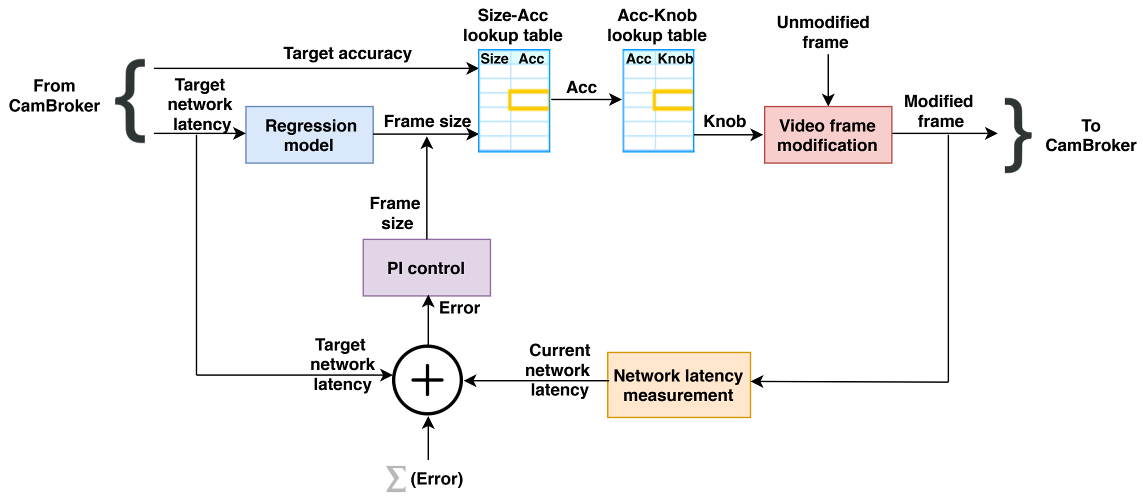


Figure 4.3: Block diagram of the network latency controller incorporated in Mez.

The video frames published at the CamBroker are sent to the latency controller through a Control API call. Note that the latency controllers on each IoT camera node operate independently of one another. The lack of centralized control allows the scaling of the IoT camera nodes.

The controller maintains application specified network latency and inference accuracy in presence of Wi-Fi channel interference using the algorithm outlined in the pseudo code shown in Algorithm 1 (Section 3.3). The network latency for different video frame sizes (see Section 3.2.2) are assumed to be available from prior characterization of the video frames in the targeted deployment environment. The application accuracy for different tuning knob settings is also assumed available for the targeted application through prior characterization. The almost linear dependence of latency on video frame size observed in Section 3.1.3 facilitates the use of linear regression model of latency on frame size.

If the application requested latency and accuracy are infeasible, the application is notified. At this point, the application has to decide whether to continue operation with relaxed latency/accuracy requirements, or notify the system operator of failure.

4.2.3 In-memory Log

The design of Mez storage is targeted to provide low latency read-write operations. We take advantage of the particular features of machine vision at the IoT edge, both to ensure high performance, and simplify the design of the storage. As shown in Figure 4.4, the storage is an in-memory log, which is an append-only, circular buffer. The log consists of $< timestamp, videoframe >$ key-value pairs stored in increasing order of timestamps. The log at the IoT camera node stores video frames either generated from a single camera, or those modified by the latency controller to satisfy latency requirements. This log is in turn replicated on demand at the Edge server. For N IoT camera nodes, the Edge server thus holds N replicated logs. Subscriber machine vision applications are served from the log at the Edge server. A machine vision application subscribes to one or more such logs. Also, one or more machine vision applications could subscribe to a particular log. The logs are hence designed to support a single writer, but multiple readers.

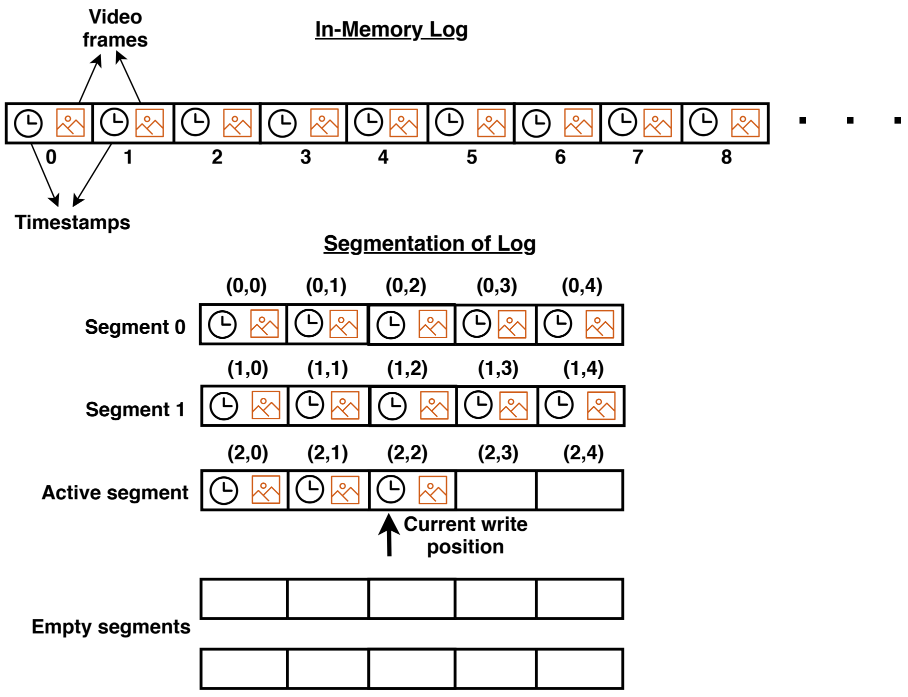


Figure 4.4: Mez in-memory log. The storage is an in-memory log, which is an append-only, circular buffer. The log consists of $< timestamp, videoframe >$ key-value pairs stored in increasing order of timestamps. Concurrent read/write performance is improved through fine grained locking by segmenting the log.

To ensure low latency, the logs only utilize the DRAM for storage. Unlike general purpose storage, there is no requirement to delete arbitrary video frames from the log. Instead, video frames with increasing time stamps are appended to the log, which wraps back when the capacity is exceeded, overwriting existing entries with older timestamps. An attempt to append a video frame with a timestamp earlier than the last entry in the log is rejected. The lack of a need to support update and delete operations, and the sorted (by timestamps) video frames in the log, simplify the design of the log and prevent memory fragmentation. Point queries are done efficiently with binary search. Range queries are also readily supported by querying the starting and ending timestamp, returning the video frames corresponding to an interval that includes the requested time range.

A 1 GB of in-memory log at the IoT camera node holds approximately 7 minutes worth of video frames (assuming 500 KB per frame, and 5 fps). The real-time machine vision applications are assumed to consume the data within this time frame. The log is persisted on the disk (in the background) only for recovery from failure (described in Section 4.2.4). Due to the possible physical insecurity of the hardware, the video frames stored on disk are encrypted at rest. The encryption/decryption, and disk accesses are relatively long latency operations, motivating the avoidance of disk access in the read/write critical path. For the short time duration the video frames are held in the DRAM, the data is assumed to be safe from illegal access due to the volatile nature of the memory.

Although the log is replicated from the IoT camera node to the Edge server, no attempt is made at the storage layer to ensure consistency of the video frames between the IoT node and the Edge server. Instead, we rely on the lower layers of the network (TCP) for accurate replication. Also, in practice we observe that the ability of machine vision applications to tolerate errors in the video frame data, allows the use of simpler transport protocols (UDP) that does not support re-transmissions. Concurrent read/write performance is improved through fine grained locking by segmenting the log. Each segment is protected with read-write locks. Note that reads can occur from many segments concurrently, while only one segment is active for write.

4.2.4 Fault tolerance

Mez is designed to recover from the following failures:

- Crashes of brokers - EdgeBroker and CamBroker
- Crash of the network latency controller at the IoT camera node
- Corruption of log segments on disk

Fault detection: Mez uses RPC timeouts to detect failures. The EdgeBroker detects failed CamBrokers through time outs on the Subscribe API call. The publisher detects failed CamBrokers through timeouts on the Publish API call. Subscriber applications detects a failed EdgeBroker through time outs on the Subscribe API call. The Cambroker detects a failed latency controller through time outs on the internal SetTarget API call. The time out duration for the Subscribe RPC depends on the video frame rate (fps), baseline wireless network latency, and if TCP is used - the re-transmit timeout. The Publish timeout is typically small since the camera and the CamBrokers are connected through a high speed interface such as USB. The Control RPC timeout is determined by the time taken by the controller to modify the video frame. We thus avoid use of explicit heartbeats, and instead take advantage of the continuous streaming of video frames from the cameras to detect component failures. For systems where the camera transmissions may be intermittent, explicit heartbeats will need to be added to monitor the health of individual components.

Fault recovery: When the subscriber application detects that the EdgeBroker has failed, it tries to reconnect with the EdgeBroker a finite (configurable) number of times or until it gets a response. As a part of the recovery process, the EdgeBroker reconstructs the logs persisted in the disk. A CRC is calculated and stored along with the on-disk log segments to detect partially written segments, which are discarded during the recovery process. The EdgeBroker then starts to accept connections from retrying subscriber applications. The CamBrokers follow a similar recovery process. Note that Mez has no inherent mechanism to restart failed brokers. Instead, Mez relies on an external service such as Kubernetes [72] to restart failed brokers (see Section 7.1).

4.3 Summary

In this Chapter, we presented Mez - a Publish-Subscribe messaging system for distributed machine vision at the IoT Edge. We first described the simple interface that Mez provides for applications to communicate their performance requirements. We then explained Mez's data model which is designed to be a simple key-value pair.

Mez's internal design consists of three architectural components - message broker, in-memory log and latency controller. Further we presented the detailed design of message brokers and in-memory log in Mez and demonstrated how the network latency controller (described in Chapter 3) can be integrated with Mez. We also list out the faults that can be tolerated by Mez and their detection and recovery mechanisms.

CHAPTER 5: EVALUATION

We evaluate the pub-sub latency performance of Mez both with scaling of peer IoT camera nodes, and with scaling the number of subscriber applications. We compare Mez with the state-of-the-art low-latency NATS [11] pub-sub messaging system. The pub-sub latency is the end-to-end time taken for a video frame to be published by the camera and subscribed by the application. The pub-sub latency includes the Publish and Subscribe API completion times, network latency, video frame modification times by controller (for Mez), and all processing delays inside the messaging system. Note that the pub-sub latency does not include the compute time for the pedestrian detection application.

The IoT Edge test bed used in the evaluation is described in Section 3.1.1. The test bed is composed of an Edge server, and five IoT camera nodes connected to the Edge server through Wi-Fi (802.11ac). All latencies are measured between one of the IoT camera nodes, and the Edge server. The workload used is the pedestrian detection application with OpenPose described in Section 3.2.3 using video frames from the JAAD and DukeMTMC datasets described in Section 3.1.1. All latencies are measured at the 95th percentile with video frames streamed from the IoT camera node to the Edge server at 5 fps. The latency is calculated as time difference $t_{Received} - t_{Send}$. The camera nodes and the Edge server are synchronized using PTP synchronization protocol [66] before the start of the measurements.

This Chapter is structured as follows:

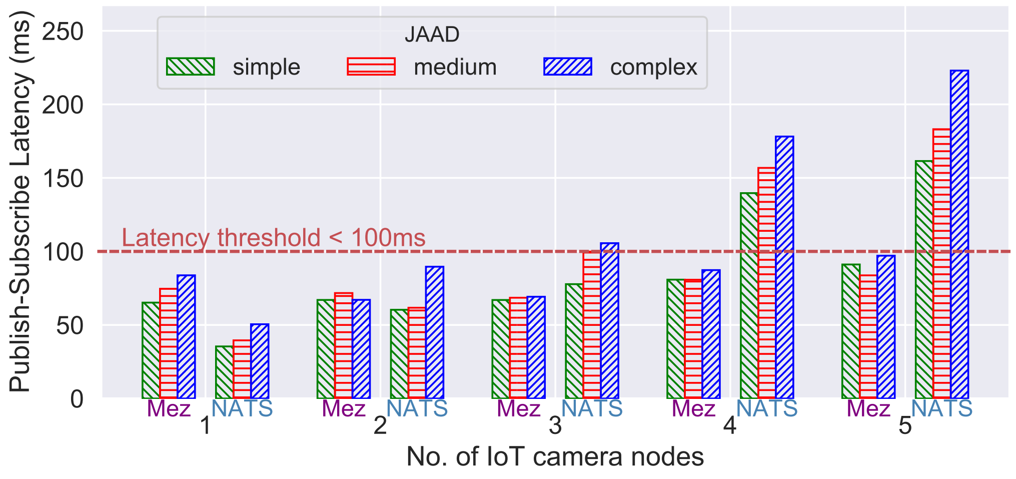
1. Section 5.1 presents pub-sub latency performance evaluation of Mez with scaling of peer IoT camera nodes.

2. Section 5.2 evaluates pub-sub latency performance of Mez with scaling of subscribers.
3. Section 5.3 compares the breakdown for different components of the pub-sub latency for Mez and NATS.
4. Section 5.4 summarizes this Chapter.

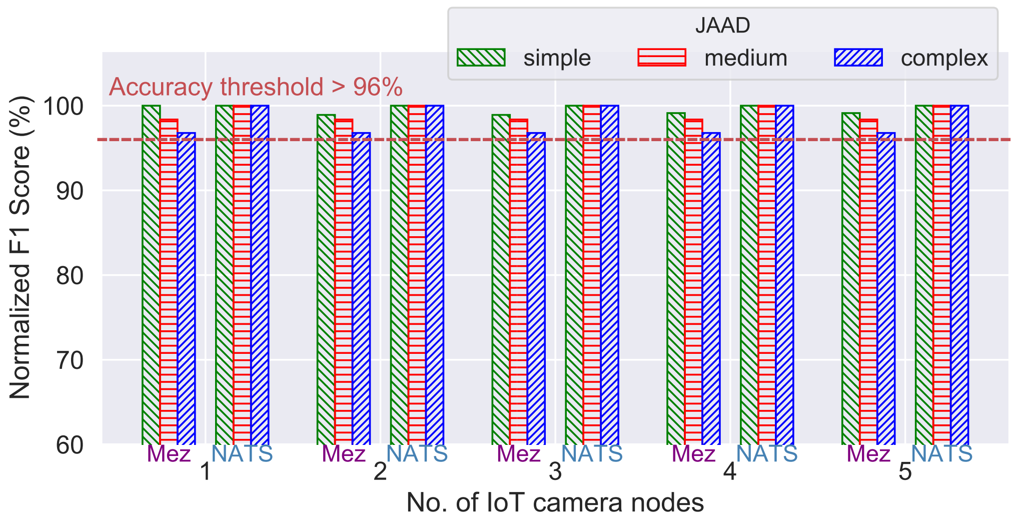
5.1 Node Scaling

Figure 5.1a shows the per frame pub-sub latency for Mez and NATS for the JAAD dataset as the number of IoT camera nodes is scaled from 1 to 5. The latency and the normalized $F1$ accuracy thresholds are set at 100 ms, and 96% respectively. This setup emulates a scenario where a single subscriber (for example, a machine vision application for object re-identification across multiple camera views) requests video frames from multiple IoT camera nodes. As seen from Figure 5.1a, when the number of IoT nodes are increased, Mez is able to maintain the pub-sub latency under 100ms. However, the pub-sub latency of NATS shows a super-linear increase with IoT node scaling, since NATS does not perform any type of network latency control. Figure 5.1b shows the accuracy (normalized $F1\%$) achieved by Mez and NATS as the IoT camera nodes scale. Since NATS always sends unmodified video frames, it maintains the maximum accuracy for all cases. Mez shows a worst case accuracy reduction of 3.3%.

Similar evaluation is performed on the DukeMTMC dataset with the latency threshold set at 100ms for video frames with simple, medium and complex scene dynamics. Since the median video frame size for DukeMTMC is greater than 1 MB, we could not evaluate NATS due to its 1 MB message size limit. When the number of IoT nodes is scaled from 1 to 5, Mez is able to maintain the pub-sub latency under 100ms (Figure 5.2a) while maintaining the accuracy reduction less than 4.2%(Figure 5.2b).

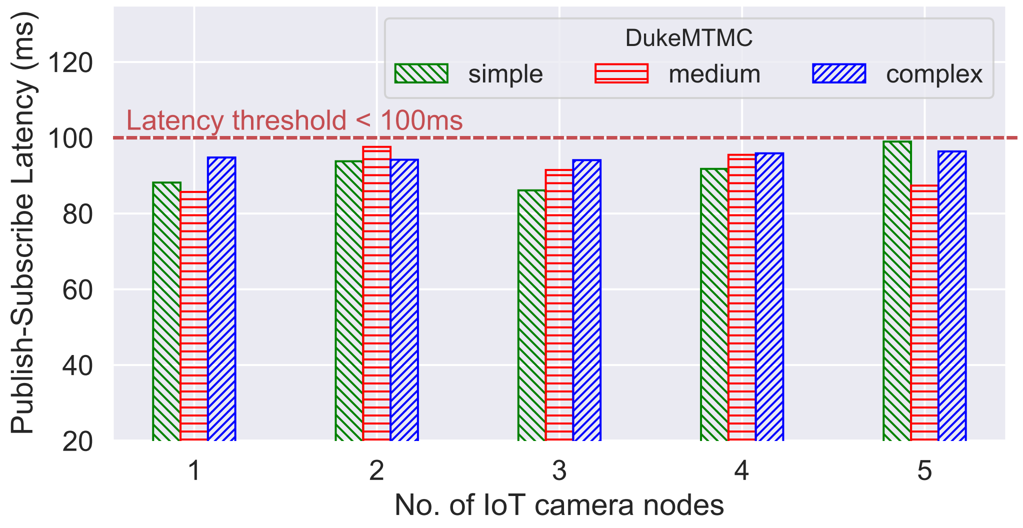


(a)

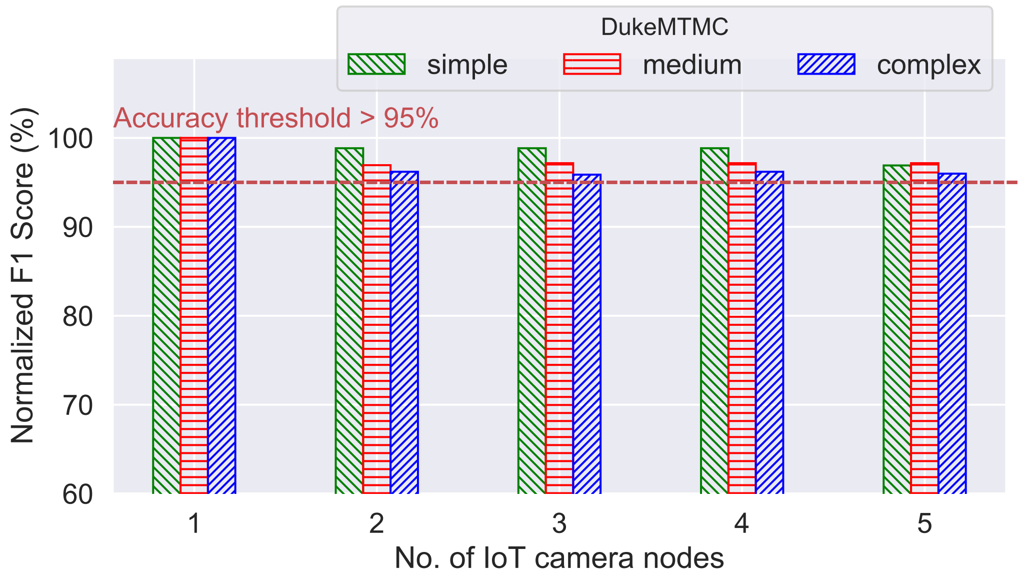


(b)

Figure 5.1: Pub-sub latency (95th percentile), and pedestrian detection accuracy with IoT node scaling for Mez and NATS for JAAD dataset with simple, medium and complex scene dynamics video frames. In (a) Y axis shows the per frame Publish-Subscribe latency. In (b) Y axis shows the accuracy in terms of the normalized F1 score percentage. X axis of both figures indicates the number of IoT camera nodes. Unlike NATS, Mez is able to achieve the latency threshold of 100ms as the number of IoT camera nodes scale. The resulting loss of accuracy is less than 3.3%.



(a)



(b)

Figure 5.2: Pub-sub latency (95th percentile), and pedestrian detection accuracy with IoT node scaling for Mez for DukeMTMC dataset with simple, medium and complex scene dynamics video frames. In (a) Y axis shows the per frame Publish-Subscribe latency. In (b) Y axis shows the accuracy in terms of the normalized F1 score percentage. X axis of both figures indicates the number of IoT camera nodes. Mez is able to achieve the latency threshold of 100ms as the number of IoT camera nodes scale. The resulting loss of accuracy is less than 4.2%. Since NATS has a 1MB message size limit, DukeMTMC frames cannot be sent/received using NATS.

5.2 Subscriber Scaling

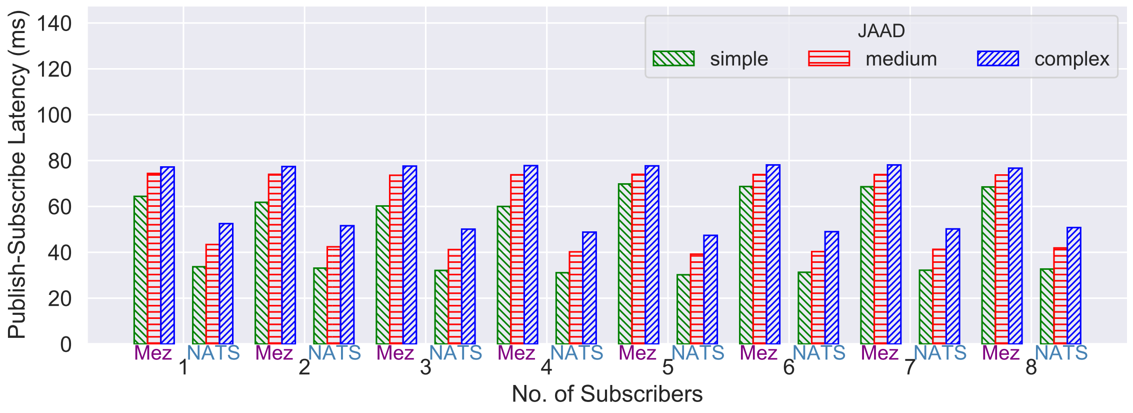
Figure 5.3 shows the pub-sub latency for Mez and NATS as the number of subscribers are scaled. Poor subscriber scaling would indicate concurrency limitations. This set up emulates the operational scenario at the IoT-Edge where multiple vision applications (subscribers) request video frames from a single camera. In this case, since only a single IoT camera node (to which the producer is publishing video frames) is operational, there is no channel interference due to peer IoT camera nodes. Both Mez and NATS scale well as the number of subscribers are increased from 1 to 8 with minimal degradation in latency. However, Mez has a higher latency than NATS due to controller overheads.

5.3 Pub-Sub Latency Breakdown

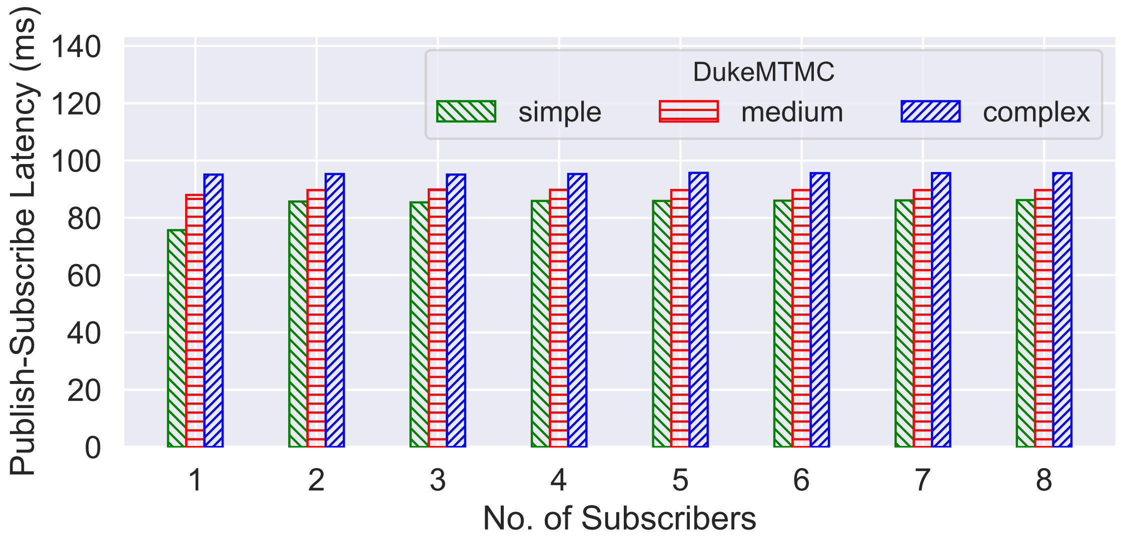
Figure 5.4 shows the breakdown for different components of the pub-sub latency for Mez and NATS with all 5 IoT camera nodes transferring video frames to the Edge server. The measurements are taken for complex scene dynamics video frames from the JAAD dataset. For NATS, the network latency dominates the overall latency at 96.2%. For Mez also, the network latency is the dominant component at 65.7%, with the controller overhead being the next highest at 20.5%. About half the controller processing time is due to the video frame modification, with the video frame copying between the logs at IoT camera nodes accounting for the remaining time. Use of GPUs available on the Nvidia Xavier boards to perform the video frame modification can potentially lower the video frame processing time. Integrating the controller as a part of the CamBroker instead of the current approach of the controller as a separate microservice, could result in lowering the video frame copying overheads.

5.4 Summary

In this Chapter we evaluated Publish-Subscribe messaging system, Mez on an IoT Edge test bed with five IoT camera nodes and an Edge server. The workload used for

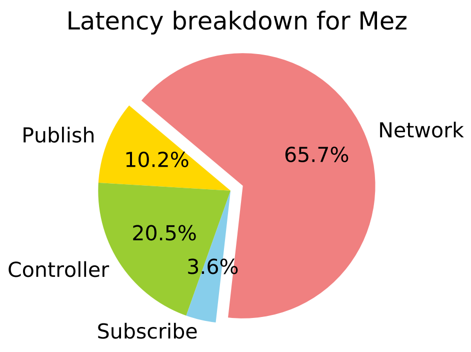


(a)

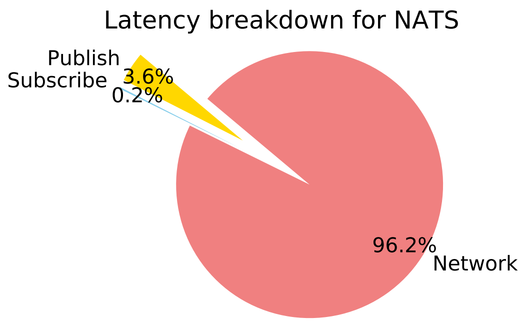


(b)

Figure 5.3: Subscriber scaling for Mez and NATS for (a) JAAD and (b) DukeMTMC datasets with simple, medium and complex scene dynamics. Y axis shows the per frame Publish-Subscribe latency and X axis shows the number of subscribers.



(a)



(b)

Figure 5.4: End-to-end latency breakdown for (a) Mez and (b) NATS in presence of 4 peer nodes. The measurements are taken for complex scene dynamics video frames from the JAAD dataset with median frame size of 970KB, streamed at 5fps rate. For NATS, the network latency dominates the overall latency at 96.2%. For Mez, the network latency is the dominant component at 65.7%, with the controller overhead being the next highest at 20.5%

evaluation is the pedestrian detection application with OpenPose using video frames from the JAAD and DukeMTMC vision datasets. We compared Mez with state-of-the-art low latency NATS messaging system. Our experimental evaluation indicates that, when IoT camera nodes are scaled to five, Mez is able to maintain the pub-sub latency under application requested threshold, while NATS shows a super-linear increase in pub-sub latency. However, both Mez and NATS scale well upto eight subscribers with minimal degradation in pub-sub latency.

CHAPTER 6: SCALING THE NETWORK LATENCY CONTROLLER

Chapter 3 presented the design of the network latency controller in Mez that dynamically adjusts the video frame quality to satisfy latency and machine vision application inference accuracy requirements. The latency controller adjusts video frame quality using a set of frame quality modification techniques called tuning knobs (explained in Section 3.2.1). To facilitate the video frame quality tuning process the latency controller implements a lookup table that stores candidate knob settings, resulting video frame size (after modifying frames by applying tuning knobs) and inference accuracy values (by feeding modified frames to machine vision application). The entries in the lookup table are then used by the controller to search for the tuning knob combination that satisfy both the latency and inference accuracy demands.

A lookup table style of design is adopted for the controller since the total combinations of settings for 5 knobs (identified in Section 3.2.1) results in 1250 values, a small number easily stored in memory. However exhaustively combining more number of knobs with more fine-grained knob settings results in combinatorial explosion of the knob search space. In this case, computing and storing the resulting frame size and machine vision inference accuracy for all the knob combinations become prohibitively expensive.

In this Chapter we propose two scalable algorithms to solve the combinatorial explosion problem - design space pruning heuristic algorithm, and machine learning based algorithm. This Chapter is structured as follows:

1. Section 6.1 lists 10 video frame quality modification techniques (tuning knobs) and their knob settings that reduce video frame size. Among these, first 2 techniques are already introduced in Section 3.2.1. In this Section, we have

identified additional knob settings for these 2 tuning knobs.

2. Section 6.2 first presents the equation to quantitatively evaluate total number knob combinations, given the number of knobs and knob settings. We also propose two scalable algorithms (pruning heuristic and machine learning based) to determine the settings of the tuning knobs that simultaneously satisfy video frame size, and inference accuracy requirements.

3. Section 6.3 presents extensive experimental evaluation of the algorithms for object detection machine vision application using the EfficientNet [18, 17] Deep learning architecture on the Microsoft COCO 2017 [19] dataset.

4. Section 6.5 concludes this Chapter.

6.1 Additional Video Frame Quality Tuning Knobs

In this section we explore additional video frame quality modification techniques which when applied on video frames reduce the frame size. Further, we investigate the impact of the reduced frame size on machine vision application inference accuracy. The techniques, referred to as tuning knobs, are described below.

1. **Knob1 - Colorspace modifications:** Video frames can be converted from one colorspace to another resulting in total size reduction. We select 8 such colorspace modifications which are BGR↔Gray, BGR↔XYZ, BGR↔HSV, BGR↔HLS, BGR↔LAB, BGR↔LUV, BGR↔YUV and BGR↔CrCb. Our choice of color space modifications can reduce the video frame size by as much as 60%.

2. **Knob2 - Blurring:** Video frames can be blurred by passing them through various low pass filters. We chose the filter kernel sizes of 5, 8, 10, 15 and 20 as possible knob settings. Blurring the video frames can reduce the video frame size by as much as 75%.

3. **Knob3 - Denoising:** Noise content in video frames can be removed by passing them through denoising filters. The 5 knob settings selected for this knob are denoising filter strengths of 3, 10, 15, 20 and 30. Denoising knob can reduce the frame size upto 67%.

4. **Knob4 - Contrast stretching:** Contrast stretching can be done on video frames by performing range normalization over the frame pixel array. The norm values for range normalization are chosen from 0.3 to 0.9 with intervals of 0.1. Contrast stretching using these knob settings can achieve upto 70% size reduction.

5. **Knob5 - 2D filtering:** The 2D filtering approach convolves a video frame with a kernel and removes the noise in the frame. This knob could reduce the frame size as much as 68% when filter kernel sizes of 5, 6, 7 and 8 are applied on the video frames.

6. **Knob6 - Gaussian filtering:** A video frame can be convolved with a Gaussian kernel to remove Gaussian noise from the video frame. Selected kernel sizes are 5, 11, 21, 31 and 51 and achieved size reduction is 79%.

7. **Knob7 - Median filtering:** The median filtering technique computes the median of all the pixels under a kernel window and the central pixel is replaced with this median value. This technique is highly effective in removing salt-and-pepper noise from video frames. By choosing the knob settings as 5, 9, 11, 13 and 19, frame size reduced upto 72%.

8. **Knob8 - Bilateral Filtering:** Bilateral filtering removes noise in video frames while preserving the sharp edges in them. For this knob the filter sizes are chosen to be 10, 30 and 50 and filter sigma values are chosen as 70, 150 and 200. Upon application of this knob video frames reduced in its size by 64%.

9. **Knob9 - Erosion:** Erosion is a type of morphological transformation that erodes away the boundaries of objects in video frames and is useful in removing small white noises. Choosing erosion filter kernel sizes to be 5, 10 and 15 could achieve size reduction of 62%.

10. **Knob10 - Dilation:** Dilation is another type of morphological transformation which is the opposite of erosion. This knob dilates video frames using a kernel structure. The kernel structure sizes chosen are 5, 8 and 10 with size reductions of upto 52%.

Note that each of these knobs can be set independent of the other knobs potentially resulting in a large (≈22 million) search space.

6.1.1 Impact of Tuning Knobs on Inference Accuracy

Reducing the information content by applying these tuning knobs on video frames could impact the object/event detection inference accuracy of machine vision applications consuming these video frames. In general, the impact is dependent on the particular machine vision application. We choose object detection as the machine vision application since it is widely used, and is a basis of other computer vision tasks such as object tracking, and activity detection. The object detector EfficientDet [17] is based on EfficientNet [18], a deep learning neural network developed by Google brain team. It achieves state-of-the-art accuracy while being up to 9x smaller than competing models and using significantly less computation. Microsoft COCO [19] is a publicly available vision dataset consisting of 91 object categories (classes) with a total of 2.5 million labeled instances in 328K images. We input the original and modified video frames from COCO 2017 dataset to EfficientDet to generate object detections on video frames with bounding boxes drawn around the objects.

To evaluate these detections, each ground truth bounding box for that frame (publicly available) is matched exclusively to the outputted bounding box based on highest

Intersection over Union (IoU) overlap. Positive matches with an IoU greater than a threshold are considered True Positives; result bounding boxes without ground truth matches are considered False Positives; and each unmatched ground truth box is considered a False Negative. These records are utilized for mAP (Mean Average Precision) calculation.

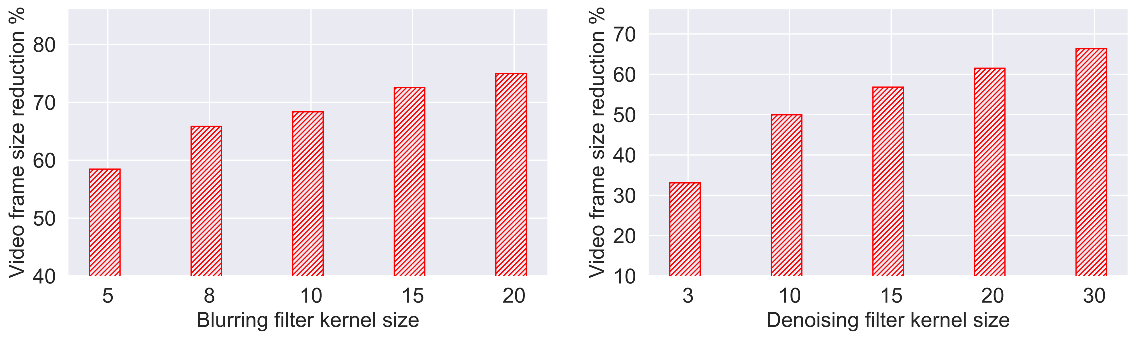
For object detection, we utilize the mAP metric with an Intersection-over-Union (IoU) threshold of 0.5. Equation 6.1 defines the calculation for mAP. Precision is $\frac{TP}{(TP+FP)}$, where TP, FP, and FN are the number of True Positives, False Positives, and False Negatives respectively. Average Precision calculated at a single IoU threshold (0.5 in our case) for a single object class is denoted as, $AP^{IoU=.5}$. Finally, mAP is obtained by averaging $AP^{IoU=.5}$ over different classes in the chosen dataset.

$$mAP = \frac{1}{\#classes} \sum_{class \in classes} AP^{IoU=.5}[class] \tag{6.1}$$

where $\#classes$ represents number of classes of objects in the video frames and $AP^{IoU=.5}[class]$ represents $AP^{IoU=.5}$ for a specific object class.

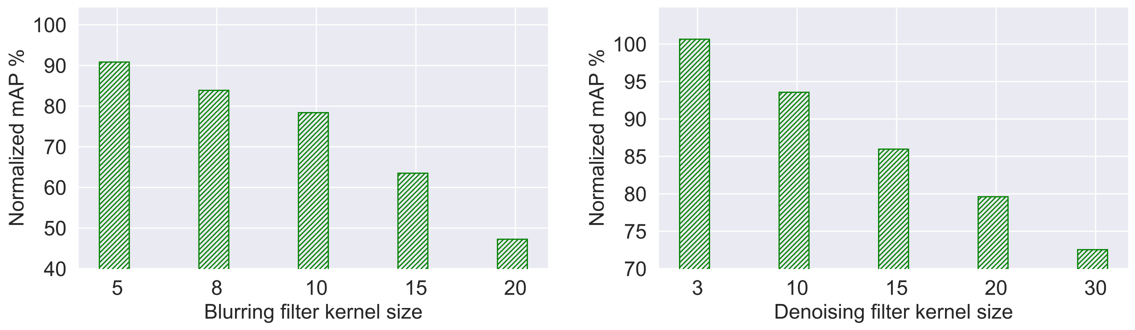
Figures 6.1 and 6.2 show the impact of application of tuning knobs, blurring and denoising (Knobs 2 and 3) on and video frame size and mAP. The mAP of unmodified (without the application of any tuning knob) video frame is considered as the baseline mAP. The mAP values obtained after applying tuning knobs on video frames are normalized with respect to the baseline mAP. This characterization corresponds to mAP generated (using EfficientDet-D0 [73] model) on 300 images chosen from COCO 2017 dataset.

Figures 6.1 and 6.2 demonstrate that the application of these tuning knobs can reduce the video frame size as much as 75% and application inference accuracy upto 47% (of unmodified video frames). A similar trend for video frame size reduction and accuracy degradation can be observed for other tuning knobs (Knobs 1 and 4-10) as well.



(a) (b)

Figure 6.1: Characterization of the impact of application of tuning knobs, (a) blurring (knob 2) and (b) denoising (knob 3) on video frame size. Blurring and denoising knobs reduced the frame size as much as 75% and 67% respectively.



(a) (b)

Figure 6.2: Characterization of the impact of application of tuning knobs, (a) blurring (knob 2) and (b) denoising (knob 3) on mAP (application inference accuracy). Blurring and denoising knobs achieved a normalized mAP score of 47% and 72% (of baseline mAP) respectively.

From Figure 6.2, note that setting 1 of knob 2 reduced the mAP to 90.8% of the baseline and setting 2 of knob 3 reduced mAP to 93.5% of baseline. Figure 6.3 shows the visual impact of application of setting 1 of knob 2 and setting 2 of knob 3 on object detections. Because of the effect of quality modification, both the video frames have missed detections for a few objects. But note that both video frames still preserve most of the true detections for object categories such as person and bicycle.

These observations indicate that approximate computing exploited in Chapter 3 (in case of tuning knobs explored in Section 3.2) is also applicable in case of the

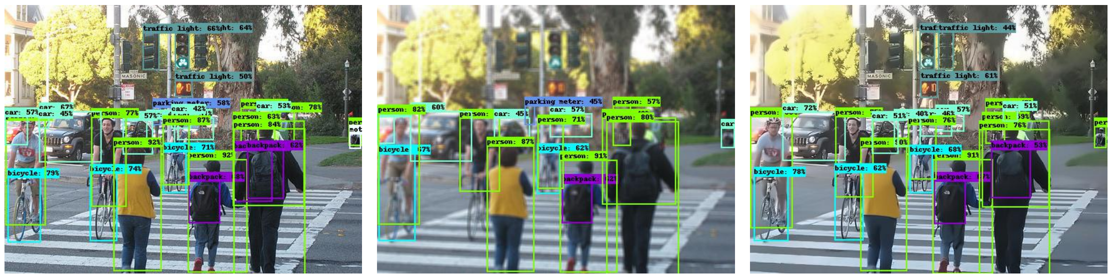

(a) (b) (c)

Figure 6.3: Visual impact on object detections before and after the application of tuning knobs. (a) Unmodified video frame, video frame after applying (b) blurring knob with a kernel size of 5 and (c) denoising knob with a kernel size of 10. Both video frames (b) and (c) have few missed detections, but still preserve most of the true detections for object categories such as person and bicycle.

additional tuning knobs described in this Section. Lower quality video frames with smaller size, can be transferred with lower network latency (from IoT camera node to Edge server) if it provides acceptable machine vision application inference accuracy. Therefore these additional knobs can also be used as potential tuning knobs to design the network latency controller (Chapter 3) that adjusts frame size by combining multiple tuning knob settings. But exhaustively combining all knob settings of the additional knobs identified, results in a huge knob design space (≈22 million). Therefore characterizing the frame size and inference accuracy for all tuning knob combinations takes a great deal of time. In the next Section we present two scalable approximate computing algorithms to identify useful knobs (knobs with reduced frame size and acceptable accuracy) by eliminating the need for exhaustive characterization of the total knob design space.

6.2 Scalable Approximate Computing Algorithms

In this Section, first we quantitatively represent the knob design space using an equation, then we describe the scalable approximate computing algorithms for knob selection.

6.2.1 Knob Search Space

The observations from previous section indicate that tuning knobs provide a mechanism to send reduced size video frames with acceptable inference accuracy. It should be noted that the machine vision application specifies the inference accuracy and network latency bounds of the video frames that need to be transmitted. Video frames satisfying requested network latency can be found if we have the size of these frames pre-characterized (from the network latency vs. video frame size linear relationship described in Section 3.1.3).

To select the knob settings that constitute the targeted frame size and target inference accuracy metrics, all combinations of settings of identified knobs need to be characterized. However, combining the knob settings for different tuning knobs results in a combinatorial explosion of the design space as seen in Equation 6.2. For tuning knobs from 1 to n, this number can be represented using the equation,

$$k_{total} = k_1 \times k_2 \times ... \times k_i \times ... \times k_n \tag{6.2}$$

where k_{total} represents the total number of knob combinations and k_i represents number of knob settings for the the i^{th} knob. For the 10 tuning knobs identified in Section 6.1 this results in a total number of 22,394,880 knob combinations. Computing and storing the resulting frame size (by applying knob combinations sequentially on video frames) and computing the machine vision inference metric (by feeding the modified video frames to the machine vision application) become prohibitively expensive.

We therefore explore two scalable algorithms to solve the combinatorial explosion problem - the design space pruning heuristic algorithm, and machine learning based algorithm.

6.2.2 Pruning Heuristic Algorithm for Knob Selection

We use pruning heuristic algorithm to successively filter out knob combinations that result in lower performance on the inference metric. In this algorithm we consider the k_{total} knob combinations of Equation 6.2 as the heuristic decision space of the problem.

The algorithm works as follows. The search space consists of n tuning knobs with each tuning knob i consisting of k_i tuning knob settings. In the first step of this algorithm we change the knob setting for a single tuning knob (while keeping settings of other knobs at their defaults) and calculate the frame size and inference metric. We repeat this process for all the knob settings independently for each knob. We then filter out knob settings that results in a low performance on the inference metric (lower than a application specified threshold). This completes the first step of the algorithm. In the second step, we choose pairs of knob combinations from distinct knob combinations obtained from the first step. A filtering step similar to the first step is then applied to weed out knob combinations with lower performance on the inference metric. This completes the second step of the algorithm. The process is repeated next considering 3 distinct knob combinations from the knob setting obtained from step 2. The algorithm terminates when all n distinct knob combinations are considered in the n^{th} step. Each step i, prunes the design space by eliminating low performing knob combinations.

The algorithm for the pruning heuristic knob selection is outlined in the pseudo code shown in Listing 2.

6.2.3 Machine Learning Algorithm for Knob Selection

In contrast to pruning the entire knob space to identify useful knobs using the pruning heuristic algorithm, a machine learning algorithmic approach can be used predict video frame size and inference accuracy. To build the model, we first collect

Algorithm 2: Pruning heuristic algorithm

```
Result: Video frame quality knob settings
InferenceAccuracy = $Iacc$ ;
InferenceAccuracyThreshold = $Iacc_{Thres}$ ;
numKnobs = $n$ ;
step, $i = 1$;
while $i <= n$ do
    from n knobs, choose i settings = $nC_i$ ;
    for each setting $k_i$ in $nC_i$ do
        apply setting $k_i$ on video frame;
        calculate video frame size;
        calculate $Iacc$ on video frame;
        if $Iacc < Iacc_{Thres}$ then
            discard setting $k_i$;
    end
    increment step, $i$ by 1
end
```

the input data required to train the model. We sample a small subset of knob combinations from the total possible set of knob combinations, and evaluate the video frame size and machine vision application inference accuracy for the sampled knob combinations. We then develop a machine learning model and train it using the sampled knob data to predict the size and inference accuracy for remaining knob combinations. The detailed steps of the algorithm are described below.

The first step of the algorithm is the data collection process. Here the data represents tuning knob combinations identified using Equation 6.2, the resulting video frame size after applying tuning knobs, and the associated inference accuracy of machine vision application. Since the the knob sample space is large, calculating the frame size and inference accuracy for all the knob combinations is a time consuming process. Therefore we select a representative set of samples from the knob search space using a sampling method. A Latin Hypercube Sampling (LHS) [74, 75] method can be used to generate near-random samples from the multi-dimensional search space of knobs. LHS with a 'maximin' criteria is specifically chosen to avoid the correlation between samples by maximizing the minimum distance between samples. [76].

The next step is to modify the video frames by applying the sampled knob combinations, and recording the resulting video frame sizes. The modified video frames are fed to the machine vision application to generate the inference accuracy for each knob combination. At the end of this step for each knob combination we have a frame size and inference accuracy value.

We aim to two build two models - one to predict frame size and another to predict inference accuracy values; we consider the knob combinations as common input features to both models. We note that all the input features (knob combinations) have their dependent variable values (frame size and inference accuracy) labeled. Hence we conclude that a supervised learning method needs to be used for this kind of labeled data samples. Another observation is that both the dependent variables are real-valued quantities that need to be predicted. Therefore a regression model would be best suited for this type of data. Also we observed that knobs have a non-linear dependence on both frame size and inference accuracy. Additionally, video frames with same size map to different inference accuracy values. Due to the complex dependence of knobs on frame size and inference accuracy, we chose to proceed with a non-linear machine learning model to estimate the dependent variables.

We experimented with multiple non-linear models such as polynomial regression, Support vector machine [50], Random forest regression [51] and Decision tree regression [52]. But none of the models could provide sufficient level of prediction accuracy. Then we explored a recently proposed model called CatBoostRegressor [57] from open source gradient boosting library CatBoost [58] because of the categorical nature of the input features (knob combinations). Models from CatBoost library outperforms state of the art gradient boosting libraries such as XGBoost [56] and LightGBM [59] in terms of model quality and training speed. CatBoost is a decision tree based library that uses two phases to predict the next tree. In CatBoost the second phase is performed using traditional Gradient Boosting Decision Tree (GBDT) [55] scheme and for the first phase a modified version of GDBT is used.

6.3 Implementation and Results

In this Section we present the experimental evaluation of the scalable approximate computing algorithms described in Section 6.2.

6.3.1 Pruning Heuristic Algorithm

We implemented all the additional knobs identified in Section 6.1 using open source computer vision library OpenCV [67]. The video frames were chosen from Microsoft COCO 2017 dataset. A set of 300 video frames belonging to object classes person, bicycle, car and traffic light were selected for the experimental evaluation. We chose the object detector model EfficientDet [17] (based on EfficientNet [18]) as the machine vision application to evaluate its inference metric (mAP) on modified video frames. EfficientDet model D0 was chosen among models D0 to D7 because it is least complex model (has low number of model parameters and low computation latency) and is suited for resource constrained Edge devices.

A set of 146 knob combinations was identified using the pruning heuristic algorithm from the 22 million knob search space explained in Section 6.2.1. To identify these knob combinations we set the inference accuracy threshold (mAP) of the EfficientDet object detector to be >80% of the baseline mAP. Note that the application of all 146 combinations of the knobs identified above result in different sizes of video frames, all lower than the original.

Figure 6.4 shows the plot of the normalized mAP expressed as a percentage vs. video frame size for video frames (from COCO dataset) modified using the filtered knob combinations. The video frame size buckets in Figure 6.4 corresponds to different combinations of the knob settings with different resulting mAP. Note that higher mAP indicates higher inference accuracy for EfficeintDet. (The reason for size bucket 100-120 showing a max normalized mAP greater than 100% is because of the effect of

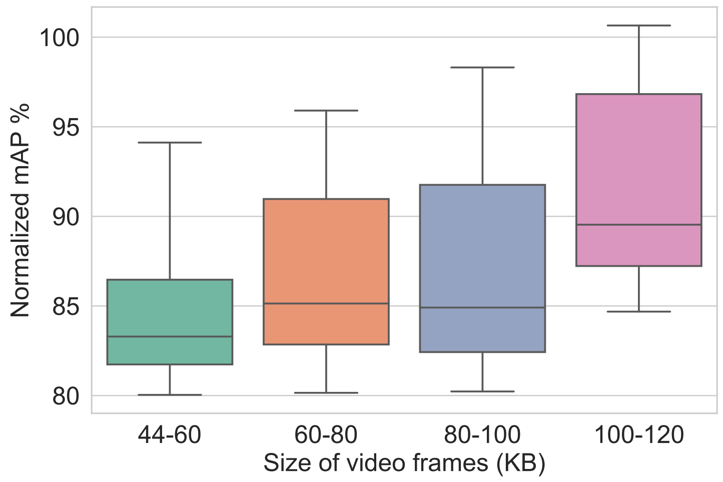


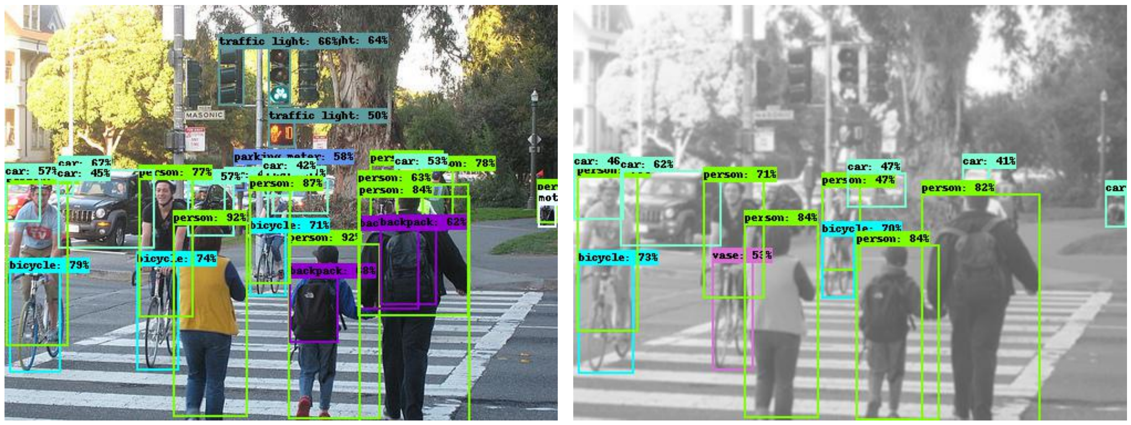
Figure 6.4: Normalized mAP expressed as a percentage for EfficientDet mAP for video frames from COCO dataset. Note that each video frame bucket corresponds to different combinations of the knob settings with different resulting mAP.

knob setting 1 of knob3-denoising, which actually caused the mAP to become slightly higher than baseline mAP.)

Using the pruning heuristic algorithm we could identify knob settings that achieved 71.33% video frame size reduction with mAP score of as much as 82.37% of the baseline mAP. This key observation enables transmission of smaller sized images with less mAP from the IoT camera nodes to the Edge server in presence of Wi-Fi channel interference.

Figure 6.5 shows the visual impact of application of tuning knobs (colorspace modification, denoising, contrast stretching and gaussian filtering) resulting in a normalized mAP of 82.37% and size reduction of 71.33%. We note that the object detections in Figure 6.5b contains most of the true detections (from unmodified video frame) except a few missed detections such as a bicycle, a parking meter, one car and two traffic lights. Two wrong detections resulting here are a bicycle detected as vase, and a person detected as car.

Using Equation 6.2 we estimate the total number of knobs need to be evaluated to be 354 by executing the pruning heuristic algorithm (Listing 2) for steps from 1 to 4 and using the 10 knobs and their settings identified in Section 3.2.1. The application set inference accuracy threshold was set to be >80% of the baseline mAP. The computation time taken to evaluate mAP scores (using EfficientDet-D0 on an Nvidia GeForce 1060 GPU) for 354 combinations for 300 video frames from COCO dataset is 10.03 hours. The total time taken for evaluations escalates substantially when number of knobs and/or settings for each knob increase.



(a) (b)

Figure 6.5: Visual impact on object detections before and after the application of tuning knobs. (a) unmodified video frame and (b) video frame after applying knobs - colorspace modification, denoising, contrast stretching and gaussian filtering.

6.3.2 Machine Learning Algorithm

We sampled 1000 knob combinations (using Latin Hypercube Sampling) from the knob sample space to generate the input data for the model. Next, the sampled knob combinations are applied to video frames to calculate the video frame size. All the knob settings are represented as categorical features before feeding into the model. We then divided the samples into train and test samples using an 80-20% split. As explained in Section 6.2.3, we chose the machine learning model CatBoostRegressor [57] model from open source library for gradient boosting library catboost [58]. We trained the model and predicted the frame sizes for the knob combinations from the test sample set. We obtained train and test Root Mean Square Errors (RMSEs) for this model as 0.74KB and 1.5KB respectively.

Next we take knob settings along with their video frame sizes as input features to construct a model to predict the machine vision application inference accuracy. The inference accuracy metric chosen for EfficientDet is mAP. With knob settings represented as categorical features, we use the CatBoostRegressor model to predict mAP scores for test knob combinations. For this model we achieved train and test RMSEs as 0.51% and 1.6% (normalized mAP) respectively.

Using the machine learning algorithm we could identify knob settings that achieved 71.37% video frame size reduction with mAP score of as much as 80.93% of the baseline mAP.

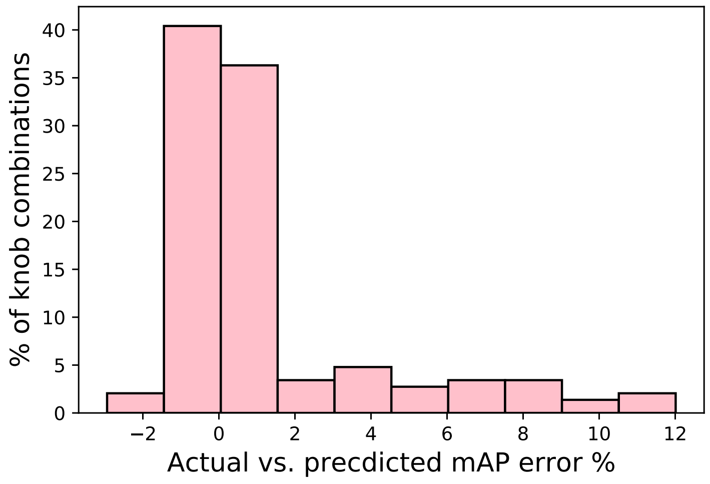


Figure 6.6: Histogram showing the distribution of error between actual and predicted mAP scores for percentage of knob combinations chosen to test the mAP ML model. 82.19% of the knob combinations have predicted mAP error variation of only ±3% of actual mAP.

Figures 6.6 and 6.7 show the percentage variation of actual and predicted mAP and video frame size with respect to percentage of knob combinations in test sample. Video frame size histogram (Figure 6.7) shows 78.08% of the knob combinations fall within ±10% of video frame size error. The mAP histogram (Figure 6.6) shows that 82.19% of knob combinations fall within ±3% of mAP error. Since we could predict accurately (with less than 10% error) most of the knob combinations' video frame size and inference accuracy using the constructed models, we conclude that the models are sufficiently accurate. The computation time taken to evaluate mAP scores (using EfficientDet-D0 on an Nvidia GeForce 1060 GPU) for 1000 knob combinations (used for training and testing the models) for 300 video frames from COCO dataset is 28.33 hours.

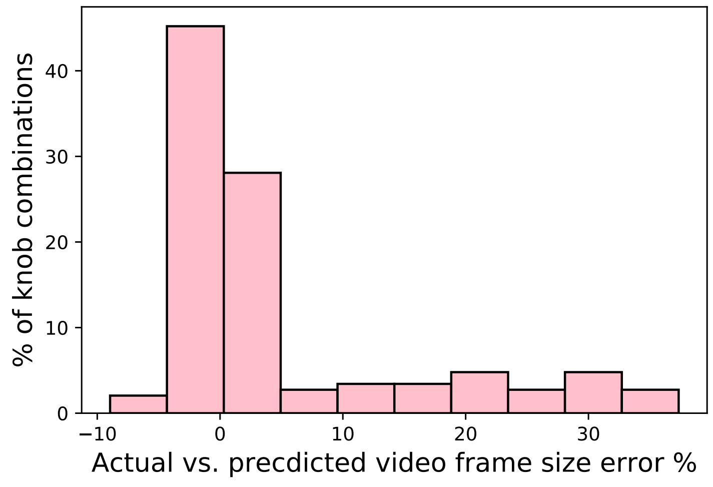


Figure 6.7: Histogram showing the distribution of variation between actual and predicted video frame size for percentage knob combinations chosen to test the video frame size ML model. 78.08% of the knob combinations have predicted frame size error variation of only ±10% of actual frame size.

Comparing the two scalable approximate computing algorithms, the Categorical boost machine learning model based algorithm achieves comparable accuracies to the pruning heuristic algorithm (within ±3% error for mAP model, and ±10% for video frame size model). The machine learning approach is more scalable both in terms of the number of knobs, and the number of settings for each knob, since it only has a one time training cost.

6.4 Discussion

The machine learning models could be evaluated in real-time to predict the inference accuracy, and frame size resulting from a given knob combination. Here, the machine learning models act as objective functions that need to be minimized or maximized in a multi-objective optimization space [77]. The goal would be to find a set of solutions (knob combinations) as close as possible to satisfying the conflicting objectives. The resulting knob combinations can then be used to cached in a lookup table when the controller is operated in the real time for fast access.

A multi-objective optimization problem can be expressed in a general form as shown in Equations 6.4.

Find the vector,

$$X^* = [{x_1}^*, {x_2}^*, ..., {x_n}^*]^T \tag{6.3}$$

to optimize,

$$F(X) = [f_1(X), f_2(X), ..., f_k(X)]^T, \tag{6.4}$$

subject to m inequality constraints,

$$g_i(X) \leq 0, \; i = 1 \text{ to } m \tag{6.5}$$

and p equality constraints,

$$h_j(X) = 0, \; j = 1 \text{ to } p \tag{6.6}$$

where $X^* \in R^n$ is the vector of decision or design variables, and $F(X) \in R^k$ is the vector of objective functions, both of which must be either minimized or maximized.

In a multi-objective optimization problem (MOO), the goodness of a solution is determined by 'dominance'. A solution x_1 dominates x_2 if and only if, solution x_1 is no worse than x_2 in all objectives and solution x_1 is strictly better than x_2 in at least one objective. Given a set of solutions, the non-dominated solution set of a MOO is defined as the set of all the solutions that are not dominated by any member of the solution set. The non-dominated set of the entire feasible decision space is called the Pareto-optimal set. The boundary defined by the set of all points mapped from the Pareto optimal set is called the Pareto optimal front. The goal of MOO is to find a set of solutions as close as possible to Pareto optimal front.

There are various ways in which a multi-objective optimization problem can be solved. Some of the classic ways to solve MOO problems are listed below.

1. Scalarizing: In this method, the original problem with multiple objectives is converted into a single-objective optimization problem. For instance, in a linear scalarization method, a set of objectives are scalarized into a single objective by adding each objective pre-multiplied by a user supplied weight. The weight of an objective can be chosen in proportion to the relative importance of the objective. This method is relatively simple, but finding weight vectors to obtain a Pareto optimal solution in a desired region in the objective space is difficult.

2. A posteriori methods: A posteriori methods aim at producing all the Pareto optimal solutions or a representative subset of the Pareto optimal solutions. Posteriori methods fall into either one of the following two classes: mathematical programming-based a posteriori methods, and evolutionary algorithms. In mathematical programming-based a posteriori methods an algorithm is repeated and each run of the algorithm produces one Pareto optimal solution. Some of the example methods of this class include Normal Boundary Intersection (NBI) [78], Successive Pareto Optimization (SPO) [79] and Directed Search Domain (DSD) [80]. In evolutionary algorithms one run of the algorithm produces a set of Pareto optimal solutions. Examples of evolutionary algorithms are Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [81] and Strength Pareto Evolutionary Algorithm 2 (SPEA-2) [82].

6.5 Summary

In this Chapter we explored the problem of making the network latency controller scalable when the tuning knob design space is huge, resulting in combinatorial explosion. We identified additional video frame transformation techniques, that can result in reduced frame size, and effectively acts as network latency tuning knobs for machine vision applications. Since the knob design space suffers from a combinatorial explosion problem precluding exhaustive characterization of knob combinations, we investigated scalable algorithms to facilitate our approximate computing approach. We experimentally evaluated two approaches - a heuristic based pruning algorithm of the design space, and a Categorical boost machine learning model based algorithm. Both approaches were able to reduce the video frame size by upto 71.3% while achieving a inference accuracy of 80.9% of the inference accuracy of the unmodified video frames. The machine learning model has a fixed training cost compared to the heuristic based pruning algorithm, while being inherently more scalable.

CHAPTER 7: CONCLUSIONS

Multi-camera based Deep Learning vision applications subscribe to the Edge computing paradigm due to stringent latency requirements. However, guaranteeing latency in the wireless communication links between the IoT cameras nodes and the Edge server is challenging, especially in the cheap and easily available unlicensed bands due to the interference from other camera nodes in the system, and from external sources. In this dissertation, we show how approximate computation techniques can be used to design a latency controller that uses multiple video frame quality control knobs to simultaneously satisfy latency and accuracy requirements for machine vision applications. We also established the need for a suitable messaging abstraction at the Edge for machine vision application developers to deploy vision applications that consume video streams from one or more cameras.

We proposed the design of a publish-subscribe messaging system, Mez for distributed machine vision at the IoT Edge. Mez provides machine vision applications at the Edge the ability to specify network latency upper bound for the video frames transferred from the IoT camera nodes to the Edge server, along with an accuracy lower bound that the application can tolerate. Many machine vision applications support the approximate computing paradigm, where useful enough results can be obtained despite reduced quality input. We incorporated the approximate computing based latency controller in Mez and it achieves application specified network latency despite channel interference, by modifying the video frames to reduce their sizes such that the application accuracy specifications are satisfied as well. Additionally, the design of Mez incorporates an in-memory log based storage that takes advantage of specific features of machine vision applications to implement low latency operations.

We also discuss the fault tolerance capabilities of the Mez design.

Our experimental evaluation of Mez on an IoT Edge testbed with a pedestrian detection machine vision application indicates that Mez is able to tolerate latency variations of up to 10x with a modest drop in application accuracy of less than 4.2%. We then revisited the latency controller and made its design scalable using two approximate computing based algorithms - a heuristic based design space pruning algorithm, and a Categorical boost based machine learning algorithm. Experimental results on an object detection application on the Microsoft COCO 2017 data set, indicates that proposed methods were able to reduce the video frame size by upto 71.3% while achieving an inference accuracy of 80.9% of that of the unmodified video frames. The machine learning model has a high training cost, but has a lower inference time, and is scalable and flexible compared to the heuristic design space pruning algorithm. We also discuss alternatives to some of our design choices, and suggest directions for future work.

7.1 Discussion and Future Directions

In this Section we review the different choices made in the design of the messaging system, Mez, and discuss alternatives.

GPU computing at IoT nodes: In our work we have only used the 8 core ARM CPU available on the Nvidia Xavier board. By using the GPU available on the Nvidia Xavier board, additional computationally intensive video frame modifications including performing object detection at the IoT camera node could be employed.

Video frame compression: The video frames that are transferred from the IoT camera node to the Edge server could be compressed (for example, using H.264) to reduce the data size. However, Canel et. al. [34] suggest that a low quality H.264-encoded 1080p (1920x1080 pixels) stream is insufficient to perform accurate analysis for vision analytic applications such as traffic monitoring and pedestrian tracking.

Availability: Regarding service availability, the current version of Mez does not support replicated physical nodes (neither IoT camera nodes, nor Edge sever). For hardware failures, such as if an IoT node fails, the video frames from the associated camera become unavailable. However, if the Edge server fails, then entire system becomes unavailable. The ability to support replicated Edge server components (logs and the EdgeBroker) is part of future work. However, we note that computational resources are constrained at the Edge due to power, space, and cost considerations. Thus, physical replication of resources may not be practical on the Edge. An alternative worth investigating is the use of Cloud for fail over of the Edge server so that the system is still operational, albeit at a reduced performance.

For software failures, Mez could rely on container orchestrators such as Kubernetes [72] for resurrecting failed services. In this case, Mez needs to be containerized (for example, using Docker containers [83]), with Kubernetes orchestrating the containers. The microservice architecture of Mez allows ready containerization of the brokers, and the latency controller. A Continuous Integration/Continuous Delivery framework (for example, with Jenkins [84]) could also be used to ease the deployment and updating of Mez without loss of service. Note that Kubernetes itself will need hardware redundancy to guarantee its availability. With limited hardware, an interesting possibility is to use the Cloud to ensure availability of Kubernetes.

Security: Mez takes advantage of the security features of gRPC to implement TLS based certificate authentication for clients and servers, as well as TLS based encryption of video frames in transit. Additionally, all sensitive video frames at rest that are persisted on disk are encrypted. We leave the implementation of additional security measures such as Role Based Access Control (RBAC) for application access to video frame data, and the verification of the authenticity of container images to future work. It should be noted that a Trusted Platform Module (TPM) can be incorporated with the Edge device hardware to securely store sensitive information such as security keys and passwords.

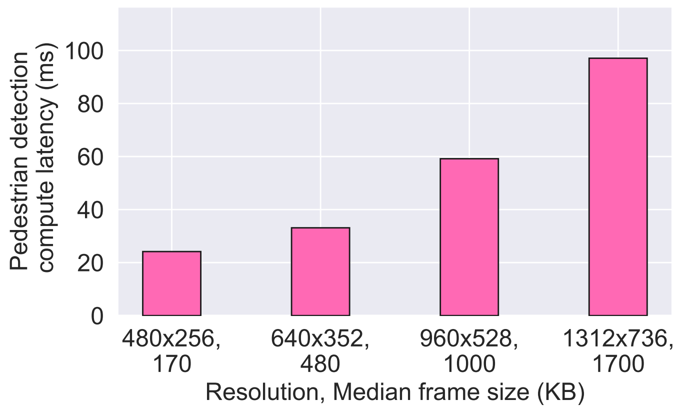


Figure 7.1: Compute latency for pedestrian detection with OpenPose (on Nvidia Titan V GPU) vs. video frame size

Compute latency: In Figure 7.1 we investigate the dependence of the compute latency on the video frame size. We note that, the compute latency is only dependent on the resolution of the images. Since one of the tuning knobs is image resolution, with reduced resolution and with reduced image size, OpenPose could achieve reduction in compute latency as well.

Figure 7.1 shows that the pedestrian detection compute latency increases with increase in resolution and image size. The measurements are done on an Nvidia Titan V GPU. In this dissertation we have only focused on the network latency. However, due to the compute intensive nature of these applications, there exists a possibility of jointly optimizing compute and network latency, so as to obtain the overall desired latency. We leave the exploration of this topic to future work.

Optimal knob selection: In Chapter 6 we investigated two approximate computing based algorithms to make the latency controller in Mez scalable - the pruning heuristic based algorithm, and machine learning model based algorithm. The latency controller uses a set of knob combinations to achieve the network latency and inference accuracy constraints specified by machine vision applications. As a part of future work, the machine learning models can be used as objective functions in a multi-objective optimization space to solve for an optimal knob combination that satisfy latency and accuracy constraints.

REFERENCES

[1] X. Wang, “Intelligent multi-camera video surveillance: A review,” *Pattern Recognition Letters*, vol. 34, no. 1, pp. 3 – 19, 2013. Extracting Semantics from Multi-Spectrum Video.

[2] E. Ristani and C. Tomasi, “Features for multi-target multi-camera tracking and re-identification,” *CoRR*, vol. abs/1803.10859, 2018.

[3] K. Zhang, J. Ni, K. Yang, X. Liang, J. Ren, and X. S. Shen, “Security and privacy in smart city applications: Challenges and solutions,” *IEEE Communications Magazine*, vol. 55, no. 1, pp. 122–129, 2017.

[4] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. Klatzky, D. Siewiorek, and M. Satyanarayanan, “An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance,” in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, SEC ’17, (New York, NY, USA), pp. 14:1–14:14, ACM, 2017.

[5] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The case for vm-based cloudlets in mobile computing,” *IEEE Pervasive Computing*, vol. 8, pp. 14–23, Oct 2009.

[6] E. A. Lee, B. Hartmann, J. Kubiatowicz, T. Simunic Rosing, J. Wawrzynek, D. Wessel, J. Rabaey, K. Pister, A. Sangiovanni-Vincentelli, S. A. Seshia, D. Blaauw, P. Dutta, K. Fu, C. Guestrin, B. Taskar, R. Jafari, D. Jones, V. Kumar, R. Mangharam, G. J. Pappas, R. M. Murray, and A. Rowe, “The swarm at the edge of the cloud,” *IEEE Design Test*, vol. 31, pp. 8–20, June 2014.

[7] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, pp. 637–646, Oct 2016.

[8] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, *Fog Computing: A Platform for Internet of Things and Analytics*, pp. 169–186. Cham: Springer International Publishing, 2014.

[9] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC ’12, (New York, NY, USA), pp. 13–16, ACM, 2012.

[10] J. Kreps, N. Narkhede, and J. Rao, “Kafka : a distributed messaging system for log processing,” in *International Workshop on Networking Meets Databases (NetDB)*, 2011.

[11] “Nats, documentation.” `https://nats.io`. Last retrieved 2019-21-10.

[12] P. Software, "Rabbitmq by pivotal." `https://www.rabbitmq.com/`. Last retrieved 2020-01-10.

[13] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh, "Openpose: Realtime multi-person 2d pose estimation using part affinity fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2019.

[14] T. Simon, H. Joo, I. Matthews, and Y. Sheikh, "Hand keypoint detection in single images using multiview bootstrapping," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4645–4653, 2017.

[15] S. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional pose machines," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4724–4732, 2016.

[16] Z. Cao, T. Simon, S. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1302–1310, 2017.

[17] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10778–10787, 2020.

[18] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," vol. 97 of *Proceedings of Machine Learning Research*, (Long Beach, California, USA), pp. 6105–6114, PMLR, 09–15 Jun 2019.

[19] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), (Cham), pp. 740–755, Springer International Publishing, 2014.

[20] W. Shi, G. Pallis, and Z. Xu, "Edge computing [scanning the issue]," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1474–1481, 2019.

[21] "Aws, explore our products." `https://aws.amazon.com/`. Last retrieved 2019-21-10.

[22] "Azure, invent with purpose.." `https://azure.microsoft.com/en-us/`. Last retrieved 2019-21-10.

[23] "Google cloud, build. modernize. scale.." `https://cloud.google.com/`. Last retrieved 2019-21-10.

[24] Y. Xiao, Y. Jia, C. Liu, X. Cheng, J. Yu, and W. Lv, "Edge computing security: State of the art and challenges," *Proceedings of the IEEE*, vol. 107, pp. 1608–1631, Aug 2019.

[25] M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. PP, no. 99, pp. 1–1, 2016.

[26] D. Sabella, A. Vaillant, P. Kuure, U. Rauschenbach, and F. Giust, "Mobile-edge computing architecture: The role of mec in the internet of things," *IEEE Consumer Electronics Magazine*, vol. 5, pp. 84–91, Oct 2016.

[27] M. Sapienza, E. Guardo, M. Cavallo, G. L. Torre, G. Leombruno, and O. Tomarchio, "Solving critical events through mobile edge computing: An approach for smart cities," in *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 1–5, May 2016.

[28] O. Vermesan, P. Friess, P. Guillemin, and S. Gusmeroli, *Internet of Things Strategic Research Agenda.* River Publishers, 2011.

[29] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, (New York, NY, USA), pp. 68–81, ACM, 2014.

[30] Y. Lu, A. Chowdhery, and S. Kandula, "Visflow: A relational platform for efficient large-scale video analytics," tech. rep., June 2016.

[31] C. Neff, M. Mendieta, S. Mohan, M. Baharani, S. Rogers, and H. Tabkhi, "Revamp2t: Real-time edge video analytics for multi-camera privacy-aware pedestrian tracking," *IEEE Internet of Things Journal*, pp. 1–1, 2019.

[32] W. Zhang, S. Li, L. Liu, Z. Jia, Y. Zhang, and D. Raychaudhuri, "Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds," 02 2019.

[33] C. Pakha, A. Chowdhery, and J. Jiang, "Reinventing video streaming for distributed vision analytics," in *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*, (Boston, MA), USENIX Association, 2018.

[34] C. Canel, T. Kim, G. Zhou, C. Li, H. Lim, D. G. Andersen, M. Kaminsky, and S. R. Dulloor, "Scaling video analytics on constrained edge nodes," *CoRR*, vol. abs/1905.13536, 2019.

[35] M. A. Ben Khadra, "An introduction to approximate computing," *CoRR*, vol. abs/1711.06115, 2017.

[36] G. Dimopoulos, I. Leontiadis, P. Barlet-Ros, and K. Papagiannaki, "Measuring video qoe from encrypted traffic," in *Proceedings of the 2016 Internet Measurement Conference*, IMC '16, (New York, NY, USA), pp. 513–526, Association for Computing Machinery, 2016.

[37] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, p. 62, 2016.

[38] C. Chen, J. Choi, K. Gopalakrishnan, V. Srinivasan, and S. Venkataramani, "Exploiting approximate computing for deep learning acceleration," in *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*, pp. 821–826, 2018.

[39] A. Ibrahim, M. Osta, M. Alameh, M. Saleh, H. Chible, and M. Valle, "Approximate computing methods for embedded machine learning," in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 845–848, IEEE, 2018.

[40] F. Betzel, K. Khatamifard, H. Suresh, D. J. Lilja, J. Sartori, and U. Karpuzcu, "Approximate communication: Techniques for reducing communication bottlenecks in large-scale parallel systems," *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, p. 1, 2018.

[41] Dynam.AI, "End to end ai solutions for your business." `hhttps://www.dynam.ai/`. Last retrieved 2020-01-10.

[42] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[43] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., 2009.

[44] A. Rasouli, I. Kotseruba, and J. K. Tsotsos, "Are they going to cross? a benchmark dataset and baseline for pedestrian crosswalk behavior," in *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pp. 206–213, 2017.

[45] E. Ristani, F. Solera, R. Zou, R. Cucchiara, and C. Tomasi, "Performance measures and a data set for multi-target, multi-camera tracking," vol. 9914, 10 2016.

[46] P. Hintjens, "0mq - the guide," 2011.

[47] I. Malpass, "Measure anything, measure everything," 2020.

[48] V. MartÃ, "Brubeck, a statsd-compatible metrics aggregator," 2020.

[49] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*. USA: Addison-Wesley Publishing Company, 5th ed., 2011.

[50] T. Evgeniou and M. Pontil, "Support vector machines: Theory and applications," vol. 2049, pp. 249–257, 01 2001.

[51] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, Oct. 2001.

[52] L. Rokach and O. Maimon, *Decision Trees*, vol. 6, pp. 165–192. 01 2005.

[53] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms.* Chapman and Hall/CRC, 1st ed., 2012.

[54] Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” *J. Comput. Syst. Sci.*, vol. 55, pp. 119–139, Aug. 1997.

[55] J. H. Friedman, “Greedy function approximation: A gradient boosting machine.,” *Ann. Statist.*, vol. 29, pp. 1189–1232, 10 2001.

[56] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, (New York, NY, USA), pp. 785–794, Association for Computing Machinery, 2016.

[57] A. V. Dorogush, V. Ershov, and A. Gulin, “Catboost: gradient boosting with categorical features support,” *ArXiv*, vol. abs/1810.11363, 2018.

[58] “Catboost.” `https://catboost.ai/`. Last retrieved 2020-12-09.

[59] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu, “Lightgbm: A highly efficient gradient boosting decision tree,” in *NIPS*, 2017.

[60] N. Corporation, “Jetson agx xavier.” `https://developer.nvidia.com/embedded/jetson-agx-xavier`. Last retrieved 2020-07-02.

[61] N. Corporation, “Nvidia titan v.” `https://www.nvidia.com/en-us/titan/titan-v/`. Last retrieved 2020-10-02.

[62] NETGEAR, “Nighthawk pro gaming xr700.” `https://www.netgear.com/gaming/xr700/`. Last retrieved 2019-13-09.

[63] S. Lloyd, “Least squares quantization in pcm,” *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[64] E. Forgy, “Cluster analysis of multivariate data: Efficiency versus interpretability of classification,” *Biometrics*, vol. 21, no. 3, pp. 768–769, 1965.

[65] T. L. Foundation, “grpc - a high-performance, open source universal rpc framework.” `https://grpc.io/`. Last retrieved 2020-23-01.

[66] J. Han and D. Jeong, “A practical implementation of ieee 1588-2008 transparent clock for distributed measurement and control systems,” *IEEE Transactions on Instrumentation and Measurement*, vol. 59, no. 2, pp. 433–439, 2010.

[67] “Opencv documentation.” `https://docs.opencv.org`. Last retrieved 2019-21-10.

[68] A. George and A. Ravindran, "Latency control for distributed machine vision at the edge through approximate computing," in *Edge Computing – EDGE 2019* (T. Zhang, J. Wei, and L.-J. Zhang, eds.), (Cham), pp. 16–30, Springer International Publishing, 2019.

[69] A. George and A. Ravindran, "Distributed middleware for edge vision systems," in *2019 IEEE 16th International Conference on Smart Cities: Improving Quality of Life Using ICT IoT and AI (HONET-ICT)*, pp. 193–194, 2019.

[70] A. George, A. Ravindran, M. Mendieta, and H. Tabkhi, "Mez: A messaging system for latency-sensitive multi-camera machine vision at the iot edge," 2020.

[71] "The transport layer security (tls) protocol, version 1.2." https://tools.ietf.org/html/rfc5246. Accessed: 2020-09-12.

[72] T. L. Foundation, "kubernetes." `https://kubernetes.io/`. Last retrieved 2020-10-02.

[73] "Efficientdet." `https://github.com/google/automl/tree/master/efficientdet`. Last retrieved 2020-12-09.

[74] M. D. McKay, "Latin hypercube sampling as a tool in uncertainty analysis of computer models," in *Proceedings of the 24th Conference on Winter Simulation*, WSC '92, (New York, NY, USA), pp. 557–564, Association for Computing Machinery, 1992.

[75] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.

[76] V. R. Joseph and Y. Hung, "Orthogonal-maximin latin hypercube designs," *Statistica Sinica*, vol. 18, no. 1, pp. 171–186, 2008.

[77] K. Deb and D. Kalyanmoy, *Multi-Objective Optimization Using Evolutionary Algorithms*. USA: John Wiley and Sons, Inc., 2001.

[78] I. Das and J. E. Dennis, "Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems," *SIAM J. on Optimization*, vol. 8, pp. 631–657, Mar. 1998.

[79] D. Mueller-Gritschneder, H. Graeb, and U. Schlichtmann, "A successive approach to compute the bounded pareto front of practical multiobjective optimization problems," *SIAM J. on Optimization*, vol. 20, pp. 915–934, July 2009.

[80] T. Erfani and S. V. Utyuzhnikov, "Directed search domain: a method for even generation of the pareto frontier in multiobjective optimization," *Engineering Optimization*, vol. 43, no. 5, pp. 467–484, 2011.

[81] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[82] M. Kim, T. Hiroyasu, M. Miki, and S. Watanabe, "Spea2+: Improving the performance of the strength pareto evolutionary algorithm 2," in *Parallel Problem Solving from Nature - PPSN VIII* (X. Yao, E. K. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. E. Rowe, P. Tiňo, A. Kabán, and H.-P. Schwefel, eds.), (Berlin, Heidelberg), pp. 742–751, Springer Berlin Heidelberg, 2004.

[83] D. Inc., "docker." `https://www.docker.com/`. Last retrieved 2020-30-03.

[84] CD.Foundation, "Jenkins." `https://jenkins.io/`. Last retrieved 2020-30-03.