Scalable Distributed Architecture for Managing Large Scale Camera Networks

A thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computer Science and Engineering by Research

by

Kunal Jain
2019111037
kunal.jain@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
June 2024

International Institute of Information Technology Hyderabad
Hyderabad, India

CERTIFICATE

This is to certify that work presented in this thesis proposal titled **Scalable Distributed Architecture for Managing Large Scale Camera Networks** by Kunal Jain has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Advisor: Dr. Suresh Purini

Acknowledgments

I would like to begin by thanking my advisor, Dr. Suresh Purini. Without his guidance, motivation, help and inputs, this thesis would not have been possible. Dr. Purini gave me freedom throughout my three year journey with him to explore different research areas, while keeping me motivated to make steady progress on our work. He taught me that it is important to not only do good quality work, but also ensure that our work is impactful and we are satisfied with the outputs we produce. I hope to carry forward many of the lessons he taught me throughout my life. I would like to thank Dr. Tejas Bodas, who has further motivated me to pursue research. While working with him, he gave me an environment where I was able to explore new ideas and be as creative as possible while tackling problems. Thanks to Dr. Purini and Dr. Bodas, I have grown not only as a researcher but as an individual too.

I would further like to thank my co-authors Dr. Ravi Kiran Sarvadevabhatla, Dr. Prabuchandran K.J. and Dr. Arun Ravindran for their help and support during my research journey, as well as Kishan and Siddhant for helping to kickstart the major projects of this thesis. They provided valuable insights and suggestions that have improved the quality of our work.

I will always cherish my journey at IIIT and all the memories it has given me. I would like to thank Samay, I was extremely lucky to have you as my roommate and share such amazing memories with you. Siddhant, thank you for being the one constant friend for more than a decade now. A special thanks to Nikhil for all his help during my research and always giving me the right advice in situations I did not know what the best thing to do was. Naman and Keshav, thank you for never saying no to any crazy plans we came up with. Aman, Eshan and Shivangi, thank you for being such amazing friends and all the long sports binging sessions we had. Koda and Adwait, I am glad for your friendships and support. I would also like to thank my friends from school, Vaibhav, Kanav, Mehul and Siddhant. Vaibhav, thank you for all the fun we had when you visited us and also having space for us at your home. Kanav and Mehul, I really appreciate the time you guys took out of your India visits to come over to Hyderabad and spend time with us, it meant a lot.

A big thanks to all of you for always being supportive and listening to everything, from rants to start up ideas. Throughout these five years, I have made unforgettable memories with you guys and could not have asked for a better set of friends.

Most importantly, I would like to thank my parents and brother. Thank you Mom and Dad for your unconditional love and support, none of this could be possible without you. I can not thank you enough for providing me with quality education and freedom to pursue whatever career I will be happiest with. Thank you Kartik for always guiding me through life and teaching me to have fun along the way.

Abstract

The network of cameras around urban cities is increasing in a manner unmanageable by current infrastructure. Video feeds in these networks requires deploying extensive Computer Vision pipelines to generate useful insights. However, poorly designed architectures lead to overspending on the infrastructure by local authorities.

This dissertation proposes a scalable distributed video analytics framework that can process thousands of video streams from sources such as CCTV cameras using semantic scene analysis. The main idea is to deploy deep learning pipelines on the fog nodes and generate semantic scene description records (SDRs) of video feeds from the associated CCTV cameras. These SDRs are transmitted to the cloud instead of video frames saving on network bandwidth. Using these SDRs stored on the cloud database, we can answer many complex queries and perform rich video analytics, within extremely low latencies. There is no need to scan and process the video streams again on a per query basis. The software architecture on the fog nodes allows for integrating new deep learning pipelines dynamically into the existing system, thereby supporting novel analytics and queries. We demonstrate the effectiveness of the system by proposing a novel distributed algorithm for real-time vehicle pursuit. The proposed algorithm involves asking multiple spatio-temporal queries in an adaptive fashion to reduce the query processing time and is robust to inaccuracies in the deployed deep learning pipelines and camera failures.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

CCTV cameras provide safety and protection for individual homes, small and big businesses through 24/7 continuous video surveillance. Governments can monitor public places, rail, and road traffic networks for the safety of the citizens and law enforcement. It is estimated that the number of CCTV cameras installed will soon cross the billion mark [1]. Many challenges are involved in storing videos streamed by these innumerable cameras and answering various spatio-temporal, batch, and real-time queries.

In this dissertation, we focus our attention on cameras installed on road networks typically owned by the federal government. They span across vast geographical regions ranging from cities to states within a country. We need to engineer an easy-to-deploy scalable, extensible, and loosely coupled hardware-software distributed infrastructure that can process the following kinds of queries.

1. **Event Detection** Events such as jumping a red traffic light and running a stop sign correspond to implicit queries which are continuously active, but require no real-time processing.

2. **Outliers Detection** Examples of outlier detection include traffic jams due to traffic lights malfunctioning or accidents. These are implicit continuous queries that require near real-time processing.

3. **Tracking and Pursuit Queries** We may want to track an entity of interest, such as a car, for example, involved in a hit and run case or kidnapping an individual. These queries can be either real-time or not. The end-user could be investigating the potential paths taken in an off-line mode, or it could be an active real-time vehicle pursuit. These are complex queries that require full-fledged algorithms that make multiple spatio-temporal queries to the underlying system in an adaptive fashion.

4. **Data Analytics** We can perform many kinds of analytics such as rush hour analysis, identification of congestion routes and intersections, etc. Further, it is possible to study correlations between vehicle traffic and air quality, accidents and weather, etc.

5. **Miscellaneous** Many other types of spatio-temporal queries and complex tasks involving multiple entities of interest can be envisaged.

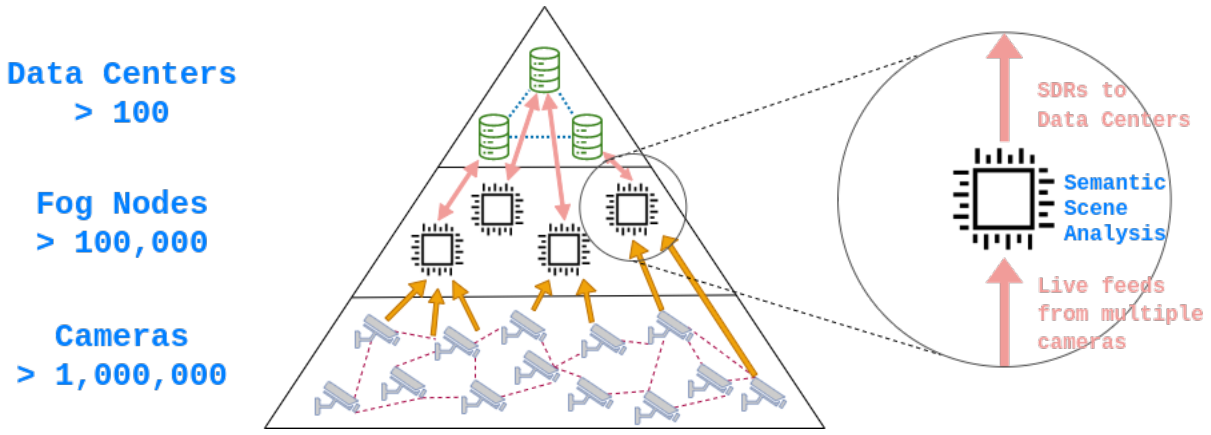


Figure 1.1: Infrastructure Architecture

1.2 Limitation of State-of-art Approaches

Few prior works [2–4] proposed cloud and data center based approaches to solve different aspects of the video analytics problem. As the number of camera feeds increases, ingesting all of them requires a correspondingly large increase in compute nodes and storage capacity. This results in big data centers and all the associated problems in managing the same. Hence, these approaches are not scalable.

Attempts have been made to solve this problem using edge and fog computing [5–11]. However, compared with our current work, they fall short in one or more of the following dimensions of the problem.

1. **Query Complexity** The spectrum of queries that can be answered as discussed in Chapter 1.1.
2. **Space and Time Granularity** Ability to answer queries at arbitrary space and time granularities (or ranges).
3. **Agnostic to Machine Learning Pipelines** The system is independent of the specific machine learning pipelines being deployed and is capable of deploying state-of-the-art models with ease.
4. **Extensibility** Easy to add new camera feeds and fog nodes to an existing system.
5. **Scalability** The system should be able to handle vast geographical regions spanning across communities, neighborhoods, suburbs, towns, cities, counties and states.

1.3 Key Insights and Contributions

Semantic Scene Analysis (SSA) involves generating a textual description of a scene through object recognition and establishing relationships between the recognized objects. This is helpful, for example, in captioning images and thereby facilitate image retrieval using textual query search. SSA is an

active challenging area of research lying at the intersection of the fields of computer vision and natural language processing. Novel deep learning architectures such as *transformers* [12, 13] are being investigated for SSA. However, these approaches are fraught with difficulties, such as the requirement of large training data and model complexity both in terms of memory and computations required. Further, the state-of-the-art still has not evolved to handle temporal relations across video frames.

Inspired by the above discussion, in this work, we propose a *cloud-fog* distributed system architecture wherein a deep learning pipeline deployed on a fog node analyzes the input video streams from the attached CCTVs and generates a sequence of *object-attributes* records in textual format. We call these records Scene Description Records (SDRs) and are transmitted to the cloud datacenter. Transmitting a textual description instead of an image itself saves network bandwidth and avoids congestion at the ingestion points of the cloud. All the query types listed in Chapter 1.1 can be answered using the database of SDRs. Complex algorithmic tasks such as vehicle pursuit, which we use as a case study to demonstrate the versatility of the proposed system, can be done using the exposed query API. Each SDR contains spatial coordinates and timestamp among other attributes. This allows us to extract spatio-temporal relations through suitably querying the database, which is an extremely difficult task even in the best deep learning based SSD systems. The stages in our deep learning pipelines involve simple tasks such as object recognition, vehicle tag extraction, vehicle color, make and model detection, etc. However, the software system architecture itself is flexible enough to deploy any complex deep learning model without restrictions. The limit of the capacity of the designed system to answer complex queries and perform rich data analytics depends on the richness of scene description and representation provided by the deployed deep learning models. For example, retrieving a vehicle using a query image requires that a compute/memory intensive vehicle reidentification CNN is deployed on the fog node. While we demonstrate the effectiveness of this framework in traffic camera networks, a similar approach is applicable across diverse domains such as surveillance networks in public places like malls and airports.

Cloud-Fog Architecture Figure 1.1 shows the proposed distributed infrastructure architecture. A few tens of spatially co-located CCTV cameras are connected to a fog node. The number of camera feeds that a fog node can process depends on the frames per second (fps), image resolution, object density per image, number and complexity of the stages in the deployed deep learning pipeline, and the compute capacity of the fog node. Many fog nodes within a geographical vicinity are connected to a mini data center. The association of fog nodes to a particular data center depends on spatial proximity and other considerations such as ownership. For example, all the fog nodes within a police precinct are connected to the corresponding mini data center [1]. Many such mini-data centers are in turn connected, resulting in what we call *multi-data centers with fog nodes* architecture. This approach has many advantages, such as the seamless addition of new mini data centers, thus facilitating the gradual expansion of infrastructure. Unlike a monolithic single data centre, a multi-data center architecture is amenable for regulatory compliance, more fault tolerance and security, and less prone to massive outages, as observed in recent

[1] A mini-data center can perhaps be called as a *cloudlet.*. Since cloudlets are usually associated with mobile devices meant for offloading computations, we stick to mini-data centers to avoid confusion.

times. By exposing well-defined API endpoints, complex analytics and query processing that require coordination from different data centers can be performed.

In the light of the above discussion on semantic scene analysis, multi-data centers with fog nodes architecture and the query types from Chapter 1.1 the following are the main contributions of this work.

1. We propose a novel approach based on semantic scene analysis to answer complex queries and perform rich data analytics. Further, the video streams are scanned and processed only once. All queries are answered from the database of scene description records (SDRs). This reduces the query processing latency by multiple orders of magnitude.

2. We propose a scalable multi-data centers with fog nodes architecture wherein the deep learning pipelines deployed on the fog nodes generate semantic scene descriptions of the input video streams. These semantic scene descriptions are transmitted to the data centers instead of videos, saving on network bandwidth, and input congestion at the data centers. This approach also reduces the resource (compute and memory) pressure on the data centers, making them manageable in terms of power and space requirements.

3. The deep learning pipelines deployed on the fog nodes are organized in the form of a directed-acyclic graph of nodes. Each node in the graph is a deep learning network by itself. The source and the destination nodes of an edge have a producer-consumer relationship connected via a queue. New nodes and edges can be dynamically added or removed to the DAG structure facilitating novel queries and analytics. The DAG structure removes redundant computation. For example, an object detection network can feed its output to multiple consumers nodes.

4. We demonstrate the versatility of our system by proposing a real-time vehicle pursuit algorithm which requires asking multiple spatio-temporal queries, and communication and coordination across multi-data centres. The algorithm can tolerate errors and inaccuracies arising out of the underlying deep learning networks and camera failures.

1.4 Experimental Methodology and Artifact Availability

We evaluate the proposed framework through a combination of experimental analysis, simulations, and analytical modeling. A cluster of five servers acts as a mini data center. The fog node software is containerized and can be deployed on any heterogeneous CPU-GPU node. Prerecorded CCTV camera feeds from the STREETS data set [14] are provided as sources for the video ingestion points on the fog nodes. We systematically benchmark the whole system's end-to-end performance and also the individual components of the system. We also evaluate the accuracy of the vehicle pursuit algorithm in the above setting. We plan to release the containerized fog node software as an artifact.

1.5 Limitations of the Proposed Approach

The stress on the fog node compute capacity increases as more complex machine learning pipelines are deployed. This is the primary limitation of the proposed approach. However, there is significant ongoing research on *AI/ML on edge* in the computer vision and machine learning community. Many research works attempt to reduce deep learning pipelines' compute and model size complexity by using techniques such as low precision arithmetic and model pruning. Hence, we envisage that the stated limitation will not be a constraining factor in the days to come and even now. In the current work, we employ different deep learning pipelines to extract scene semantics. If a single unified *scene-to-text* model can be deployed, then the fog node architecture can be made simple and the underlying hardware can be optimized for the deep learning model.

1.6 Organization

The following is the organization of the rest of the dissertation. Chapter 2 presents the related work; Chapter 3 contains the proposed overall end-to-end system architecture; Chapter 4 presents the experimental results; Chapter 5 presents the vehicle pursuit algorithm and performance results; and finally we conclude in Chapter 6.

Chapter 2

Related Work

This chapter discusses existing architectures for managing large scale camera networks. It further explains how these solutions are inadequate for the current infrastructure and discusses how this thesis overcomes those challenges.

2.1 Early Efforts

The push toward smart cities and the easy accessibility to cheap cameras has led to large-scale deployments of camera networks. These large-scale camera networks produce vast amounts of data that requires processing in order to obtain useful information that can be acted upon. There has been a lot of work in the field of distributed video analytics in recent years due to this growth in the deployment of camera networks. Early efforts [3, 4] in this direction are primarily cloud-based only, with no computations at the fog/edge. Some works, such as from Cob Parro et al. [15], use the edge computing paradigm, but the solutions are for specific problems like people detection. EdgeEye [8] and DeepFramework [16] provide a software architectural framework for deploying machine learning models on edge. VideoEdge [17] proposes a distributed query processing by constructing a query plan trading-off between resource utilization and accuracy. As discussed in Chapter 3.2, the fog node software architecture in the proposed system is so versatile that new machine learning modules can be dynamically added or removed from the existing DAG structure. The number of running instances of each DAG node can be configured, and the overall collection of instances can be scheduled for execution on CPUs and GPUs within a compute (fog) node or from across a cluster of nodes (data center).

2.2 Single Node Efforts

Nazare et al. [6] propose a research test bed system called Smart Surveillance Framework (SSF) wherein a problem can be solved by chaining a sequence of modules that communicate through data streams via shared memory. However, this is a single-node monolithic system against a scalable distributed cloud-fog architecture that can answer complex spatio-temporal queries like ours. Wei et al. [18] addressed the problem of tracking a vehicle across multiple frames within a single camera site. While

in our work, we solve the vehicle tracking problem using the CCTV camera network and the available spatio-temporal information. However, their work will complement our work by reducing the number of scene description records generated from a camera source.

2.3 Real Time Query Processing Efforts

Nasir et al. [19] proposed an approach wherein the video streams from multiple cameras within a *fog region* are processed using a Spark cluster to identify what are called as *key frames.* These key frames are transmitted to the cloud, and the sequence of keyframes constitute video summary. Our work, in comparison, uses text for video summarization, using which complex queries can be answered without revisiting the original video stream. Similar to the work of Nasir et al., as mentioned earlier, our software stack on the fog node can be deployed on a single machine or a cluster of machines (fog region). Anveshak [5] proposes a novel data-flow model specifically for tracking problems in surveillance systems based on edge and fog architecture. Our approach, where we use semantic scene analysis and a database of SDRs, is fundamentally different from Anveshak, which uses no persistent storage. Further, our system can handle many complex queries apart from supporting tracking applications.

In Anveshak [5] proposes a novel data-flow model specifically for tracking problems in surveillance systems. Their system focuses on tracking human beings, in a generally smaller geographical scale than what is required for tracking vehicles. These systems have the capacity to be implemented across edge, fog or cloud resources, but they fail to provide a data storage layer and focus primarily on real-time processing of these video streams.

EdgeEye [20] and VideoEdge [9] discuss ways to configure edge and fog nodes for maximum throughput in a real time query setting. EdgeEye provides an efficient API for deploying DNNs over edge resources and specify their parameters. While it allows for deployment of a wider class of applications, it lacks in integration with data centers or central servers to perform domain-specific queries such as vehicle pursuit or traffic congestion estimation. Sassu et. al. [21] implement an expressive real-time analytics framework that is able to integrate various pipelines well into the system along with providing APIs and interfaces to access the analytics. Their system is unable to integrate a data storage layer that can take advantage of historical data collected for new methods to be applied and tested on.

2.4 High Network Bandwidth Efforts

Zhang et al. [4] propose a general cloud-based architecture and platform that can utilise both offline and batch processing in some cloud platforms and real-time in-memory processing in other platforms. This papers work is limited to processing video data efficiently with available resources and does not talk about performing analytics queries or mining video data. These cloud-based platforms do not account for the bandwidth costs of transporting such high-resolution video that is being provided by even cheap

cameras today. Edge computing solves this issue. Cob Parro et al. [15] discuss a surveillance system based on edge computing but it is a very specific use case and not a general solution.

2.5 Infrastructure Limited Efforts

Blazeit [22] implements video analytics queries using FrameQL that extends SQL for video analytics queries and provide novel optimisation for aggregation and limit queries. The query engine can automatically optimise queries using its query optimiser and execution engines. Similarly, FALCON [11] provide a system that does video retrieval based on SSA. The SDR are computed using custom units using distributed in-memory computations. Both of these systems are able to tackle one part of the problem really well (analytics and inference respectively) but fail to provide an infrastructure for a complete pipeline.

2.6 Analytics Limited Efforts

In SIAT [3], they present a layered framework for video surveillance in a distributed system that can utilise both real-time stream processing and batch processing. However, this system only provides APIs for feature extraction. They perform good abstraction but it is not an end to end intelligent video surveillance platform.

Nirmalan & Gokulakrishnan [10] proposed an interesting framework to configure Hadoop/MapReduce for the task of video analytics in surveillance systems. The system is efficient in processing the video streams but the underlying limitations of MapReduce do not allow the system to answer all kinds of queries.

Chapter 3

System Architecture

In this chapter, we discuss the main system architecture of our proposed solution. We first describe semantic scene analysis, which is used to generate Scene Description Records (SDR). We then discuss how these SDRs will be generated at the processing layer (edge), send over to the data storage layer (cloud) and used for analytics.

3.1 Semantic Scene Analysis on Fog Nodes

Each fog node processes video feeds from multiple CCTV cameras. Each image frame is passed through a deep learning pipeline with multiple stages connected in the form of a directed acyclic graph (DAG). Each node in the DAG corresponds to a machine-learning task such as object recognition and classification. An extracted object is passed to one or more downstream stages to infer its attributes. For example, if the object is a vehicle, then downstream stages would like to determine the vehicle tag, color, make, model, and perhaps compute a general purpose *reidentification vector*. The greater the semantic richness of the attributes, the more versatile the query processing ability will be. Each object, its attributes, the GPS coordinates of the associated camera, and the timestamp put together form a Scene Description Record (SDR). Thus a single image frame can generate multiple SDRs depending on the object density. Each fog node sends the SDRs to the associated mini-data center. Each image frame is processed only once, and depending on the query, the appropriate image or a sequence of images in the form of a video will be retrieved.

Visual querying using a source query image such as that of a vehicle under pursuit is helpful in some scenarios when compared to attribute (text) based querying. This is possible by passing the vehicle image through a *reidentification network*, which generates the corresponding vector representation. This will be included as an attribute in the vehicle SDR. When a query image is given, the closest vehicle record is retrieved from the SDR database using dot product as a similarity measure. The state-of-the-art re-identification networks are computationally expensive though. Further, the vector representation length is large compared to other attributes that only need a few bytes of memory. The particular reidentification network [23] we used in our work generates an array of 2048 single-precision floating

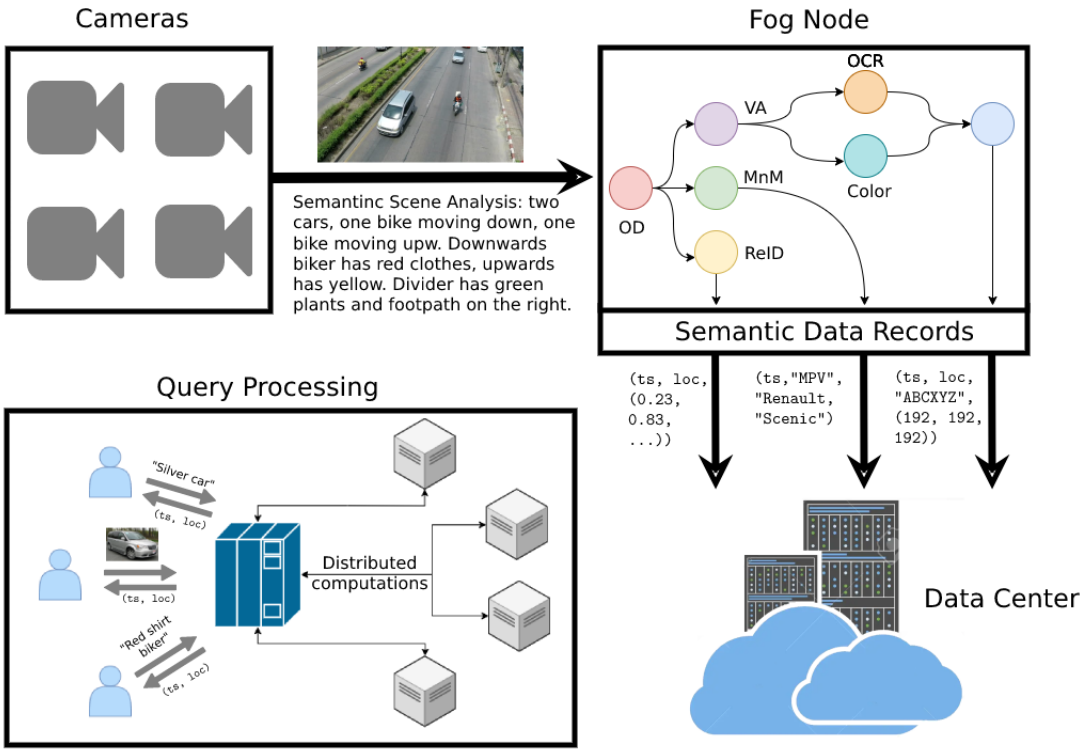


Figure 3.1: System architecture for processing video frames from cameras and answering relevant queries to the end user.

point numbers as a representation vector. This puts pressure on the communication bandwidth. Overall, there is a trade-off between the query processing ability, communication bandwidth requirement, and the compute/memory pressure on the fog nodes depending on the complexity of the deep learning pipeline deployed. The experimental results Chapter 4 presents a thorough analysis on these aspects.

3.2 Software Architecture on the Fog Node

Typically, more than one deep learning pipeline would be deployed on a fog node. For example, we could use an object detection network followed by text detection and extraction networks to detect traffic light violations. Recall that traffic light violation detection is an implicit continuous query. Another example of a deep learning pipeline is an object (for example, cars) detection network followed by a re-identification network that generates vector representations of the objects (cars). In both the above examples, the object detection network constitutes the first stage of processing. Thus, having two separate pipelines with individual object detection modules of their own will result in redundant computations. All the deployed deep learning pipelines are merged to eliminate redundant computations, forming a Directed Acyclic Graph (DAG) structure. Each node in the DAG typically corresponds to a deep learning model, while it can be any consumer-producer computation. Nodes in a DAG communi-

Architectural Component	Support
Ingestion nodes	Kafka
Distributed Deep Learning Inference	Ray
Queues	NATS Jetstream
Data storage and indexing	Geomesa on top of Accumulo
Query Processing	Spark

Table 3.1: Implementation details for various architectural components used in our system.

cate through queues. Each node can consume from multiple input queues and write to multiple output queues. Each entry in a queue is tagged with one or more subjects. Consumers of a queue can filter the entries based on the subject tags. For example, the outputs of a detection model are tagged with the corresponding object class (Ex: Car, Bike, Person, etc.). A downstream node that processes objects of a specific class, such as a car, consumes only those queue entries with the *car* tag. New nodes and edges can be added to the DAG or deleted. For example, if a novel car re-identification network has to be tested, it can be added to the DAG as a node. Then an edge from the object detection node to the newly added car re-identification node will be established.

3.3 Software Architecture on the Data Center

Each mini-data center maintains a distributed database to store all the SDRs received from the corresponding attached fog nodes. Information can often be missing from SDRs due to various reasons such a car without a number plate and as additional pipelines are added, new information can often start appearing in SDRs as well. Keeping this in mind, a NoSQL database suits our use case and provides much needed horizontal scalability.

3.4 Software Stack Selection

Table 3.1 summarizes the different components of the software stack we used in our system architecture on the cloud and the fog nodes. Figure 3.1 summarizes the overall system architecture.

3.4.1 Video Ingestion and Resource Management

Frames sent from the cameras to the fog nodes require high throughput, leading us to select Kafka for the task.

While a node represents a module in the DAG, it can have multiple instances sharing the workload i.e., parallel instances of node performing inference by consuming data from input queues to increase throughput. The computations associated with multiple node instances in the DAG are scheduled using

a Ray cluster manager. The number of instances of each DAG node and compute resources (cores, GPUs, FPGAs etc.) allocation for them can be controlled in order to avoid bottlenecks in the DAG computation flow and thus maintain steady throughput. The advantage of using Ray is that it can distribute computations to different cores and GPUs within a node. When deployed on a cluster, it can use the compute power of different nodes within the cluster. Thus the containerized software stack deployed on the fog nodes can be just deployed on a cluster node as well. This allows for a purely multi-data center approach with no fog nodes and can perhaps be adopted in an appropriate use case.

3.4.2 DAG Communication Channels

Different nodes in the DAG need to be able to communicate and get the outputs from other nodes. In order to do so, we need a publisher-subscriber model where multiple it is possible for the publisher to concurrently write to multiple channels and multiple subscribers to read concurrently from a channel. We implement this using a custom designed NATS Jetstream server.

3.4.3 Database and Indexing

As discussed in Chapter 3.3, mini-datacenters store the SDRs they recieve in a NoSQL database. In order to provide fast responses to queries, we take advantage of the fact that many of our queries will be based upon the spatio-temporal properties of the SDRs, which we use to index the rows of our database. We use Geomesa for indexing these rows upon an Accumulo database. A NoSQL database allows horizontal scalability of the system and helps us deal with scenarios where fields like number plate might be absent.

3.4.4 Query Processing

Taking further advantage of our previous software stack, we use Spark for processing queries in our database which allows us to distribute the processing of these queries as well. Our system architecture provides well-defined interfaces to permit speech and natural language based queries (Spark NLP) too.

3.5 Discussion

As the fog nodes function independently, it is possible to deploy different deep learning pipelines on them. This supports experimenting with new models on live data and phased deployment of test systems to production. One more interesting use case is population studies wherein, for example, we would like to estimate the percentage of people wearing masks. We can solve this problem by deploying a mask detection model on a subset of selected fog nodes as a sampling-based approach would suffice for this kind of study.

Chapter 4

Experimental Results

This chapter demonstrate the scalability and effectiveness of our system in extracting useful information from the cameras. We show the feasibility of our system with end-to-end simulations, analysis of network bandwidth requirements and discuss limitations as well.

4.1 Experimental Methodology

The nature of our system makes a full system experiment infeasible with the resources present to us. Thus, we evaluate our system on a level-by-level basis and provide theoretical proof of its scalability. We first evaluate how many video streams a single fog node with a certain capacity can ingest and process in near-real time. Since all the fog nodes work independently of each other, the throughput and latency analysis at a single node is applicable to all. Next, we evaluate the number of nodes that can ingest data to a single data center with our configurations and how much time it takes the data centers to process our queries for different analyses on the information received. Similar to the fog nodes, the ingestion at each mini-data center is independent of the others; thus, the results at one are applicable to all.

This process gives us a clear picture of how each part of our system would behave in a real-world deployment and the bottleneck for each level. We use these results to argue that the system would scale up to the level required to function in an entire city or state.

4.2 Semantic Scene Analysis Pipelines

The general problem of semantic scene analysis is challenging and requires hybrid vision-sequence models. So we employ simpler deep learning models and still infer rich information through complex spatio-temporal querying schemes. We experiment with several combinations of the following deep learning models. Depending on the model complexity and the number of floating point operations (FLOPS) required, we classify these models as either lightweight or heavy.

Models	Mem.	SDR/s	Latency	Estm. FLOPs
OD	1561	178.3	3847.63	6 GFLOPS
OD + ReID	3588	133.5	13885.01	54 GFLOPS
OD + OCR	1607	175.9	5323.87	12 GFLOPS
OD + OCR + CD	1707	175.7	5411.48	12 GFLOPS
OD + ReID + OCR	3633	288.5	14242.37	60 GFLOPS
OD + MnM	3831	89.6	16481.37	71 GFLOPS

Table 4.1: Comparison of memory footprint (in MB), average throughput (SDRs per second), latency (ms) and FLOPs required per inference (batched) for different deep learning pipelines on a single fog node with 10 cameras and traffic density of 18 vehicles per second.

1. *Object Detection Model (OD) :* This is a lightweight model which detects objects in an image frame, crops them and classifies them. The output of this model is typically consumed by downstream models.
2. *Reidentification Network (ReID) :* This is a heavyweight model and helps in visual query processing. We use this network in our vehicle tracking algorithm described in Chapter 5.
3. *Vehicle Tag Detection Model (OCR) :* This is a lightweight model used for vehicle number plate detection using an OCR scheme.
4. *Color Detection Model (CD) :* This is lightweight model for vehicle color detection.
5. *Vehicle Make and Model (MnM) :* This is a heavyweight model for detecting a vehicle make and model.

Using these models, we can answer a query such as *"Red Tesla around 2:00 pm near the National Museum"*. Further, we can use other attributes to enhance the accuracy of object retrieval through reidentification networks through hybrid *image-text* queries.

4.3 Model Selection and Parallel Pipelines

We compare different deep learning pipelines in terms of their capability, computations (FLOPs) required, memory footprint, latency, and throughput (SDR/s) on a fog node with a Intel Xeon Gold 6226R CPU @2.90GHz and Tesla V100 GPU. We restrict the GPU memory usage to 6 GB against the total available 16 GB and the CPU memory usage to 4GB against the total available 64GB. Ten cameras, each with 1 FPS, are connected to the fog node. The average traffic density is 18 vehicles per frame. Table 4.1 summarizes the results obtained in this setup. The table's last column only depends on the models within the pipeline and is agnostic to fps and traffic density. We can notice that the ReID and

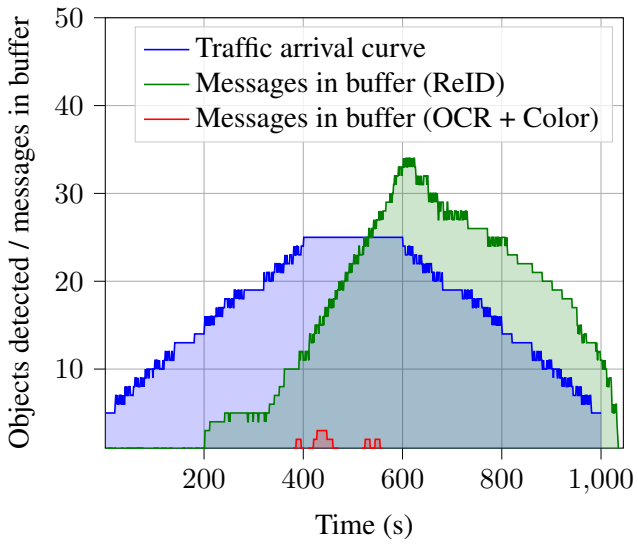
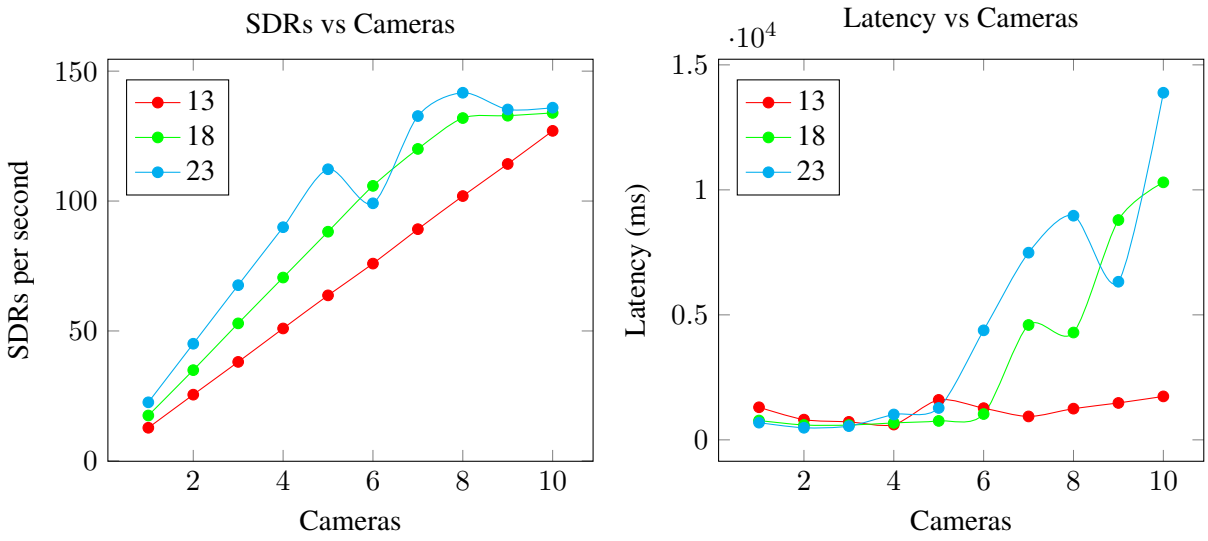


Figure 4.1: Queue size with increasing pressure from traffic

Vehicle Make/Model networks are computationally expensive and also have a 2.5x memory footprint. The time taken by the YOLOv4 object detector is negligible compared to the ReID network in the ReID pipeline. Using alternate ReID networks, which are more computationally efficient or have a shorter encoding of the input objects, would increase the pipeline's throughput. This makes the efficiency of our system directly dependent on the models the users select for their deployment.

We showcase the flexibility of our system by adding more complex parallel pipelines while experimenting on a single node. The new models are lightweight in nature, allowing our system to launched and schedule on the CPU for maximum resource utilization. The images from the OD are passed to the OCR model launched in parallel to the ReID model. The EAST [24] OCR detector captures each vehicle's number plate. A simple average of the cropped image is used to determine the color in the color detector node. These detectors work in parallel in this pipeline. The results are merged at a collection node and sent as a single SDR to the data center. This parallel pipeline is about 8% the size of the ReID pipeline and can run on a CPU. It provides an FPS of 80-90 and an average latency of 200ms. This highlights that model selection is critical to the performance of our system. Selecting models with a lower memory footprint and higher throughput while extracting similar information from the video stream should be preferred. Overall, by adding a new pipeline parallel to the previous one, a depreciation of 8-12% in the system's throughput is observed. This is due to an increase in communication time at the output queue of the OD. However, this is faster and lighter than adding a new instance of the OD itself.



(a) Vehicles processed as cameras increase

(b) Latency as cameras increase

Figure 4.2: Throughput and latency as the number of cameras varies between 13, 18 and 23.

4.4 Fog Node Analysis

We analyze the throughput of the fog node as we increase the number of associated cameras feeding at a fixed FPS of 1. We vary the average vehicles in a frame between 13 and 23 to test our system against a varying frequency of SDRs a camera generates. It also helps us analyze the backlog that might build up at the fog nodes and the time it would take to clear the queue up.

Figure 4.2 shows that the throughput of our system increases linearly until the available raw compute power is exhausted. This starts as the camera count goes beyond eight in high-traffic density scenarios. The latency increases as the computations due to the ReID model start to bottleneck the system. At this point, we can either increase the compute capacity of the fog node or use an improved ReID model, which involves fewer total operations. An alternative is to buffer the data locally and process the same when the pressure on the compute resources reduces due to a decrease in traffic density, like at night or a non-rush hour.

Figure 4.1 shows the size of the SDR buffer queue as congestion increases and puts more significant pressure on our fog nodes. We fed into the system artificially created scenes with a known frequency of cars. We ran the experiment for 10 minutes, monitoring the number of messages in the queue during that period. We notice that there is no considerable queue for the SDRs from the lightweight pipelines discussed in the following sections. The re-identification pipeline, at its peak, increases the buffer by about 35 records, each being 1MB in size. Therefore, we see an increase in buffer size at the rate of 0.1 MB/s at peak traffic. The rate at which the records buffer drops as pressure decreases is also worth noticing. Thus, in congested traffic situations, we expect the buffer to not grow beyond an acceptable

#Node	Processing time (ms)	Insertion time (ms)
1	1571	393
2	1522	404
3	1575	412
4	1546	422
5	1571	434
6	1554	442

Table 4.2: Average processing and insertion latency (ms) using multiple pipelines on a single fog node with 10 cameras and traffic density of 18 vehicles per second.

size of 750 MB even after 2 hours. This can be further improved by deploying innovative package-dropping strategies that are an active area of research right now.

4.5 Data Center Ingestion Capacity Analysis

In our system architecture, every vehicle in a video frame generates some SDRs at the fog node, which will be communicated to the data center. The number of camera feeds covered increases with the fog nodes. This puts pressure on the record insertion throughput of the distributed database at the data center. We simulate this scenario by creating dummy SDRs and ingesting them to a single data center. We stress test the system by sending records containing dummy ReID vectors. Each vector is of length 2048 with 4 bytes floating point elements, amounting to a 400 KB message size. We observe that the average insertion latency per record for 10, 20, 30, 40, 50, and 100 fog nodes are 476.45 ms, 476.45 ms, 504.31 ms, 523.30 ms, 547.23 ms, and 577.60 ms, respectively. All the fog nodes were uploading an average of 50 records per second or about 400 MBps insertion. If there are 100 fog nodes, with ten cameras per fog node, then a testbed data center with five nodes, such as the one we deployed in this work, can handle 1000 camera feeds overall.

4.6 End-to-end System Performance

While a full-scale deployment of our system is not yet possible, a small-scale deployment to test the feasibility and practicality of launching it was performed with six fog nodes and a single data center.

We examine the end-to-end performance of our system, where six fog nodes are processing data from 10 cameras each operating at 1 FPS to upload records to the data center. To test the flexibility of our system, we take fog nodes of different configurations for this experiment:

1. 2 nodes have a Tesla V100 GPU with 16 GB memory limited to 6 GB for our system and Xeon Gold CPUs with RAM usage limited to 16 GB.

2. 2 nodes have a GeForce RTX 3050 GPU with 4 GB memory and Core i7 CPU with 16 GB RAM.

3. 1 node has a GeForce GTX 1660 Ti Max-Q GPU with 6 GB memory and Ryzen 7 CPU with 12 GB RAM.

4. 1 node has a GeForce GTX 1660 Ti GPU with 6 GB memory and Core i7 CPU with 16 GB RAM.

This helps us ensure that our system is easily deployable on different kinds of hardware and is compatible with the existing infrastructure an entity might have. All the nodes are connected to the data center using a 1 Gbps link. Table 4.2 shows the average processing latency of the system as we increase the number of nodes. The results show that the performance of our system does not depreciate significantly as we increase the scale of its deployment. As the fog nodes operate independently, there is no significant variation in the processing latency. There is a slight increase in insertion time as we increase the number of insertion points, as discussed in Chapter 4.5 as well.

4.7 Query Processing

Among the many spatio-temporal queries, the most complex and compute intensive one is to find the top-k similar ReID vectors in the database against a reference vector. We use cosine similarity as a distance measure. The Geomesa database is populated with around 10^5 records randomly generated taking camera positions from the STREETS dataset [25]. We ran a query to find the top-10 records with highest co-similarity. With 10^5 records, it took an average of eight seconds to find the top-10 ReID vectors in a Spark cluster with one master and three worker nodes. Changing k between 3 and 15 with the same setup did not show significant variance in the execution duration. This is because, in all the cases, we have to evaluate cosine similarity with all vectors before sorting and picking the first k vectors. Thus the only way to improve query processing time is to filter the records using spatio-temporal constraints before the cosine similarity measure is computed. Towards this, we evaluated queries with increasing spatial-temporal ranges.

Figure 5.1a shows the time taken to process queries as we increase the bounding box size. The bounding box is a square and centered at the center of camera positions as obtained from the dataset, thus increasing the number of SDRs each query fetches. We can observe that the query time remains almost the same even as the bounding box size increases. This is because most data points are from within a smaller geographical region. However, as the time interval increases, the number of data points increases, and the query latency increases linearly.

During a vehicle pursuit, the low latency in finding the top-10 closest vector is crucial as it allows us to track vehicles in real-time once their frames have been processed and added to the database. As seen in Chapter 4.5, an average insertion time of 2 seconds and an average query time of 15 ms allow us to track vehicles within a lag of 2.5 seconds at worst.

4.8 Network Bandwidth Improvement

We deploy our system on the AI City Challenge data set to test the improvements in network bandwidth. Each camera feeds a 200-second video $\sim$180 MB video from an intersection with the vehicles staying in the field of view for a long time. A total of 13000 SDRs were observed in the original group of 10 videos due to these repetitions. Reducing the FPS to 1 reduced the number of SDRs to 631 and the size to $\sim$18 MB from each video.

We stress-test the system by sending the heavy 8 KB ReID vectors for each car over to the data center, without which the record size would be negligible. After getting processed through the fog node, 10MB of data was transmitted to the data center with the SDRs containing ReID vectors, number plates, colors and spatio-temporal tags. This shows an 18x reduction in the size of data transmitted. We also extracted helpful information in the form of SDRs in the process.

We also observed that many of the cameras had intersecting views of the road, leading to multiple SDRs for a vehicle. This raises another important aspect of the deployment of our system: the placement of the cameras and fog nodes. With an optimal placement of cameras, we can significantly reduce the number of repetitions in SDRs. Similarly, optimal placement of the fog nodes will allow us to keep the compute as close to the source of the videos as possible.

4.9 Network Choke Points

If there are c cameras per fog node and f nodes per data center, then overall, fc cameras are associated per data center. If the fog nodes are bypassed and camera feeds are streamed to a data center directly, then the ingestion points of the data center could be choked, and also the compute requirement on data center increases by a factor of fc. For example, if $c = 10$ and $f = 1000$, then from Table 4.1, we can infer that the data center requires approximately 600 TFLOPS of compute power per SDR to handle a OD+ReID+OCR pipeline. From the discussion in Chapter 4.8, we can estimate that a camera feed size per second is around 1 MB. This translates to a data center ingestion capacity of $10000 \times 1\ MB$ which is 10 GB/s. This doesn't permit the scaling of a camera network beyond some point. The proposed semantic scene analysis on cloud-fog architecture removes this limitation completely.

Chapter 5

Application - Real Time Vehicle Pursuit

In this chapter, we show how we can build a complex task such as tracking the path taken by a vehicle in real time using the proposed hardware and software infrastructure. We first summarize the deep learning pipeline for vehical re-identification on the top of which our vehicle pursuit algorithm is built. In this work, we restrict the class of vehicles to only cars.

5.1 Reidentification Pipeline

Each input image frame is passed through a YOLO object detector which extracts different objects and appropriately labels them. The car re-identification CNN [23] reads and processes the cropped images of the cars supplied by the YOLO node in the DAG via the incoming queue. The generated ReID vectors with 2048 floating-point entries are consumed by a ReID service node in the DAG. The ReID service nodes enters the ReID vectors into the database along with the spatial coordinates and the time stamp.

When a new car query image comes, then its ReID vector is computed and using a cosine similarity measure, the top k matches from the database are provided as output. In order to improve accuracy and query processing latency, while eliminating unnecessary computation, we augment the query by supplying a region of interest and time-span. Overall, a query is of the form $(\vec{R}, (lat, lon), r, T, t, k)$, which means find the top k matches for a car with ReID vector $\vec{R}$, in a region of radius r surrounding the GPS coordinates (lat, lon) in the time interval $[T - t, T + t]$. This is issued as an Apache Spark query over the underlying spatio-temporal database. This naive approach is not tolerant to inaccuracies arising out of camera angles, weather conditions, and the inherent limitations of the deployed car re-identification CNN. So, we develop an iterative algorithm robust to these faults by using the map information in addition.

5.2 Formulation and Naive Greedy Approach

We model the road network as a directed graph G where each node represents a camera. An edge exists from camera u to camera v if can travel from u to v on a road with no other camera in between.

Let I_u denote the set of nodes which have outgoing edges to node u. Let R_q denote the ReID vector associated with the query image. The confidence with which we say a car is at vertex u at time t is denoted as C_u^t. We simultaneously keep track of a set of k cars, denoted T_k^i, in which the target car is present with highest confidence, at time instant t. This set keeps evolving with every time step and the goal is to always have the target car in this maximum likelihood set.

5.3 Iterative Approach A

Let α_u^t denote the maximum cosine similarity obtained at node u and time step t with respect to the normalized query vector R_q. In other words, if A is the set of normalized ReID vectors from the database corresponding to the camera u and a time interval defined by t, then

$$\alpha_u^t = \max_{v \in A} \{v \cdot R_q\}$$

where $A \cdot R_q$ denotes a dot product operation. The likelihood of the target car being present at node u, denoted by S_u^t, depends on the estimate provided by the ReID network, which is α_u^t, and the confidence scores for the predecessor nodes in the previous time instant. If

$$\hat{C_u^t} = \max_{i \in I_u} C_i^{t-1}$$

then

$$S_u^t = \alpha_u^t + \hat{C_u^t}.$$

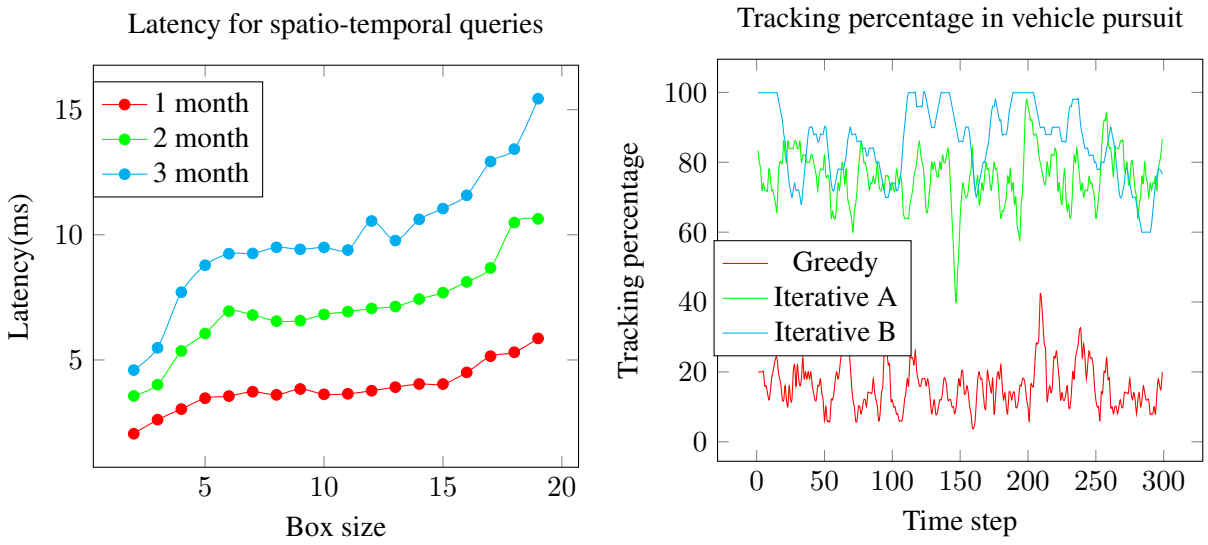
Since we know the ground truth at time step 0, the time at which we want to start tracking the vehicle, we start tracking by assigning C_l^0 as 1, where l is the location of the vehicle and the rest of the confidence as 0. From here, we take the nodes with the top k scores, which become part of the maximum likelihood set T_k^t. We normalize the confidence scores as follows for the next iteration to continue.

$$C_u^t = \frac{S_u^t}{\sum_{v \in T_k^t} S_v^t} \text{ if } u \in T_k^t$$

and $C_u^t = 0$ otherwise. This normalization allows us to give preference to tracking vehicles on nodes with high confidence neighbours.

5.4 Iterative approach B

In the previous approach, if the ReID score α_u^t at a node u falsely go up, then the confidence score also goes up even if the confidence scores of the predecessors are small. The false high value for ReID score could be due to a faulty camera or inherent limitations in the ReID network to handle certain cases. Hence, to make our system robust and tolerant to such faults, we arrive at the following method



(a) Processing time for queries as time and space parameters change

(b) Percentage of runs the vehicle under pursuit is in the top $k = 8$ maximum likelihood set T_k^t.

Figure 5.1: Analysis of our query system and vehicle pursuit algorithm

for calculating the intermediate score:

$$S_u^t = \hat{C}_u^t + \hat{C}_u^t \times \alpha_u^t$$

This method normalizes the ReID score using the previous time step confidence scores from the predecessor nodes. This avoids the situation of a car magically appearing in the radar of the vehicle pursuit algorithm.

5.5 Results

We evaluated the accuracy of our vehicle pursuit algorithm by simulating with 100 cameras for 300 steps. To test our fault tolerance, we assume the cameras flag a car false 20% of the time. Figure 5.1b shows the percentage of runs, out of 10, in which the target car is in the maximum likelihood set T_k^t for each approach described in Chapter 5. k is set to 8 for the iterative approaches and 10 for the greedy approach. When the naive greedy approach is used, we see that we are not able to track the car for more than 40% of the time steps. For iterative approach A, we can track the car between 60% and 90% of the time. This shows that taking into account the confidence levels at the previous iteration greatly increases our chances of having the target car in the maximum likelihood set. We further see that iterative approach B improves upto 10% using the new method to calculate confidence scores. This illustrates how our iterative approach can handle queries over a large period of time.

Chapter 6

Conclusion and Future Works

There is a proliferation of CCTV camera networks across various domains and applications. However, inferring useful and actionable information from this voluminous data is still challenging. At the same time, the fields of computer vision and natural language processing are making rapid strides in solving the semantic scene description problem. Using semantic scene description models, image and video retrieval tasks can be reduced to text retrieval systems. We use this observation to deploy these models on fog nodes in cloud-fog architecture for camera networks. While the state-of-the-art models are not powerful enough to capture spatio-temporal semantics, we address this problem by allowing for complex spatio-temporal queries. This approach can solve several retrieval and data analytics problems in a unified framework while optimizing on latency. From the system architecture perspective, this reduces the resource requirement on the data center side, like the compute and memory requirements and network ingestion capacity. The architecture inherently provides for scalability and extensibility. While most of the discussion in this work is focused on road camera networks, the same principles and architecture are just as applicable in other domains, and we would like to explore them in future.

List of Related Publications

1. **Jain, Kunal**, K. S. Adapa, K. Grover, R. K. Sarvadevabhatla, and S. Purini, "A cloud-fog architecture for video analytics on large scale camera networks using semantic scene analysis," in 2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid), 2023, pp. 513–523.

2. S. Jain, **Jain, Kunal**, A. Ravindran, and S. Purini, "Seer: Resource Efficient Multi-Camera Vehicle Tracking" *Under review at IEEE IoT Journal*.

6.1 Other Publications

1. **Jain, Kunal**, K. J. Prabuchandran, and T. Bodas, "Bayesian optimization for function compositions with applications to dynamic pricing," in Learning and Intelligent Optimization, M. Sellmann and K. Tierney, Eds. Cham: Springer International Publishing, 2023, pp. 62–77.

2. N. Chandak, S. Goel, **Jain, Kunal**, and A. Dasgupta, "Modelling and optimizing the allocation of covid-19 swabs to labs," Mixed Integer Programming (MIP) Workship, 2021.

Bibliography

[1] L. Lin and N. Purnell, "A world with a billion cameras watching you is just around the corner," The Wall Street Journal, Dec 2019. [Online]. Available: https://www.wsj.com/articles/a-billion-surveillance-cameras-forecast-to-be-watching-within-two-years-11575565402

[2] M. A. Hossain, "Framework for a cloud-based multimedia surveillance system," International Journal of Distributed Sensor Networks, vol. 10, no. 5, p. 135257, 2014. [Online]. Available: https://doi.org/10.1155/2014/135257

[3] M. A. Uddin, A. Alam, N. A. Tu, M. S. Islam, and Y.-K. Lee, "Siat: A distributed video analytics framework for intelligent video surveillance," Symmetry, vol. 11, no. 7, 2019. [Online]. Available: https://www.mdpi.com/2073-8994/11/7/911

[4] W. Zhang, L. Xu, P. Duan, W. Gong, Q. Lu, and S. Yang, "A video cloud platform combing online and offline cloud computing technologies," Personal and Ubiquitous Computing, vol. 19, no. 7, p. 1099–1110, Sep. 2015. [Online]. Available: http://dx.doi.org/10.1007/s00779-015-0879-3

[5] A. Khochare, A. Krishnan, and Y. Simmhan, "A scalable platform for distributed object tracking across a many-camera network," IEEE Transactions on Parallel & Distributed Systems, vol. 32, no. 06, pp. 1479–1493, jun 2021.

[6] A. C. Nazare Jr. and W. R. Schwartz, "A scalable and flexible framework for smart video surveillance," Computer Vision and Image Understanding, vol. 144, pp. 258–275, 2016, individual and Group Activities in Video Event Analysis. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314215002349

[7] L. Cheng, J. Wang, and Y. Li, "Vitrack: Efficient tracking on the edge for commodity video surveillance systems," IEEE Transactions on Parallel & Distributed Systems, vol. 33, no. 03, pp. 723–735, mar 2022.

[8] P. Liu, B. Qi, and S. Banerjee, "Edgeeye: An edge service framework for real-time intelligent video analytics," in Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking, ser. EdgeSys'18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1–6. [Online]. Available: https://doi.org/10.1145/3213344.3213345

[9] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in 2018 IEEE/ACM Symposium on Edge Computing (SEC), 2018, pp. 115–131.

[10] R. Nirmalan and K. Gokulakrishnan, "An intelligent surveillance video analytics framework using nact-hadoop/mapreduce on cloud services," Distributed and Parallel Databases, vol. 39, no. 4, pp. 873–889, Dec 2021. [Online]. Available: https://doi.org/10.1007/s10619-020-07320-z

[11] M. N. Khan, A. Alam, and Y.-K. Lee, "Falkon: Large-scale content-based video retrieval utilizing deep-features and distributed in-memory computing," in 2020 IEEE International Conference on Big Data and Smart Computing (BigComp), 2020, pp. 36–43.

[12] S. Herdade, A. Kappeler, K. Boakye, and J. Soares, Image Captioning: Transforming Objects into Words. Red Hook, NY, USA: Curran Associates Inc., 2019.

[13] S. He, W. Liao, H. R. Tavakoli, M. Yang, B. Rosenhahn, and N. Pugeault, "Image captioning through image transformer," in Computer Vision – ACCV 2020, H. Ishikawa, C.-L. Liu, T. Pajdla, and J. Shi, Eds. Cham: Springer International Publishing, 2021, pp. 153–169.

[14] C. Snyder and M. Do, "Streets: A novel camera network dataset for traffic flow," in Advances in Neural Information Processing Systems, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/ee389847678a3a9d1ce9e4ca69200d06-Paper.pdf

[15] A. C. Cob-Parro, C. Losada-Gutiérrez, M. Marrón-Romera, A. Gardel-Vicente, and I. Bravo-Muñoz, "Smart video surveillance system based on edge computing," Sensors, vol. 21, no. 9, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/9/2958

[16] A. Sassu, J. F. Saenz-Cogollo, and M. Agelli, "Deep-framework: A distributed, scalable, and edge-oriented framework for real-time analysis of video streams," Sensors, vol. 21, no. 12, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/12/4045

[17] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in 2018 IEEE/ACM Symposium on Edge Computing (SEC), 2018, pp. 115–131.

[18] P. Wei, H. Shi, J. Yang, J. Qian, Y. Ji, and X. Jiang, "City-scale vehicle tracking and traffic flow estimation using low frame-rate traffic cameras," in Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers, ser. UbiComp/ISWC '19 Adjunct. New York, NY, USA: Association for Computing Machinery, 2019, p. 602–610.

[19] M. Nasir, K. Muhammad, J. Lloret, A. K. Sangaiah, and M. Sajjad, "Fog computing enabled cost-effective distributed summarization of surveillance videos for smart cities," Journal of Parallel and Distributed Computing, vol. 126, pp. 161–170, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731518308402

[20] S. Benninger, M. Magno, A. Gomez, and L. Benini, "Edgeeye: A long-range energy-efficient vision node for long-term edge computing," in 2019 Tenth International Green and Sustainable Computing Conference (IGSC), 2019, pp. 1–8.

[21] A. Sassu, J. F. Saenz-Cogollo, and M. Agelli, "Deep-framework: A distributed, scalable, and edge-oriented framework for real-time analysis of video streams," Sensors, vol. 21, no. 12, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/12/4045

[22] D. Kang, P. Bailis, and M. Zaharia, "Blazeit: Optimizing declarative aggregation and limit queries for neural network-based video analytics," Proc. VLDB Endow., vol. 13, no. 4, pp. 533–546, 2019. [Online]. Available: http://www.vldb.org/pvldb/vol13/p533-kang.pdf

[23] L. He, X. Liao, W. Liu, X. Liu, P. Cheng, and T. Mei, "Fastreid: A pytorch toolbox for general instance re-identification," arXiv preprint arXiv:2006.02631, 2020.

[24] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang, "East: An efficient and accurate scene text detector," 2017. [Online]. Available: https://arxiv.org/abs/1704.03155

[25] C. Snyder and M. Do, "Data for streets: A novel camera network dataset for traffic flow," 2019. [Online]. Available: https://databank.illinois.edu/datasets/IDB-3671567