Towards Efficient and Accurate Visual Data Analytics System

Satyam Jay

Entry Number: 2023ANZ8224

Supervisor: Prof. Abhilash Jindal

18 March, 2024

Comprehensive report submitted to the

Amar Nath and Shashi Khosla School of Information Technology, Indian Institute of Technology, Delhi, India

Indian Institute of Technology, Delhi, India

Contents

Abbreviations

BN	Bayesian Network
CNN	Convolutional Neural Network
DAG	Directed Acyclic Graph
DL	Deep Learning
DNN	Deep Neural Network
DoG	Difference of Gaussians
DoH	Determinant of Hessian
ETL	Etract-Transform-Load
FPP	Flexible Physical Plan
GNN	Graph Neural Network
LoG	Laplacian of Gaussian
MAB	Multi Armed Bandit
ML	Machine Learning
MOT	Multi-Object Tracking
NN	Neural Network
OCR	Optical Character Recognition
QO	Query Optimization
RDBMS	Relational Database Management System
ReID	Re-identification
RNN	Recurrent Neural Network
RPS	Requests Per Second
SLA	Service Level Agreement
SOT	Single-Object Tracking
UDO	User Defined Operator
VAQ	Visual Analytics Query
VDAS	Visual Data Analytics System
VLM	Vision Language Model

Abstract

In the context of rapidly expanding digital video data from sources such as surveillance cameras, mobile devices, and social media platforms, traditional manual methods of video analysis are increasingly infeasible due to the sheer volume and complexity of the data. This has led to several research opportunities in this field. This work explores the research and development of advanced video analytics systems in recent years. We investigate the underlying techniques that enable these systems, such as optimizing machine learning inference, query accuracy estimation and heterogeneous deployment. After studying the relevant research work we come up with research problems that we can solve and contribute to the field. The research problems are related to query optimization, query synthesis and runtime adaptation.

1 Introduction

In recent years, the field of video analytics has experienced significant advancements, driven by the increasing availability of digital video data and the rapid development of computational techniques. Video analytics involves the automated extraction of meaningful information from video, transforming raw videos into actionable insights. This technology has far-reaching applications across various domains, including surveillance, healthcare, retail, transportation, and entertainment.

The primary motivation for this research work lies in the exponential growth of video data generated from various sources such as security cameras, mobile devices, and social media platforms. Traditional methods of manual video analysis are no longer feasible due to the sheer volume and complexity of the data. Therefore, there is a pressing need for efficient, scalable, and intelligent video analytics solutions that can process large datasets in real-time and provide accurate information in a timely manner.

In this work; we study various tasks and techniques involved in existing video analytics systems. Broadly we observe the following patterns in the recent works:-

- **Filtering**:- the data before applying a more expensive Machine Learning (ML) operation greatly improves latency and reduces cost. This is typically done using proxy models.
- **Indexing**:- Some metadata can be extracted during ingestion time that can later be used at query time, that also improves the query latency.
- **Query Accuracy Estimation**:- Composition of different ML operators produces different accuracy, hence during query optimization we must understand the interaction of these operators with one another in terms of their combined accuracy. This is typically done by profiling.
- **Heterogeneous**:- deployment (edge-cloud) is crucial in real-time use cases. And placement of operators become a focal concern in these cases.
- **Real-time Analytics**:- We also observe a trend towards real-time analytics, from a ingest now and query later systems. This is mainly due to two reason:- firstly, due to the progress in modern hardware we can run costly operators on affordable hardware. Secondly, due to large amount of data being generated by countless cameras installed in various settings.

Observations from this study helped us define the following research problems:-

- **Research Problem 1:** Estimating end-to-end accuracy of a query.
- **Research Problem 2:** Automatic query synthesis based on domain specification.
- **Research Problem 3:** Automatic runtime adaption of a running query.

The remaining part of this report is organized as follows: In section 2, we provide detailed background on components of a video analytics system, and its use cases. In section 3, we list down important works that paints the landscape of current state of the art. In section 4, we provide the details of the research problems that we will be working on.

2 Background

In this section, we present background of Visual Data Analytics System (VDAS). We will cover three key areas: 1) the essential characteristics of VDAS, 2) various types of commonly utilized operators in Visual Analytics Query (VAQ), and 3) practical use cases of VDAS.

2.1 Visual Data Analytics System

We characterize a Visual Data Analytics System as a platform that offers several key capabilities to its users. First, it enables users to declaratively construct a VAQ. Second, it provides a suite of commonly used operators, reducing the need for users to develop everything from scratch. Third, it includes a runtime environment for executing VAQs efficiently. Optionally, the system can also perform build-time optimization, adapt dynamically during runtime, and offer a distributed runtime environment.

A Visual Data Analytics System can also support real-time analytics, in which video is streaming from a source, analytics is performed and results are available at real-time. With advances in ML optimized hardware, real-time analytics also desirable.

A general design of VDAS is given in Figure2.1. The developer writes a logical query using the operators either provided by the system or defined by the user himself. The raw data can be ingested prior to the query execution, or it can come from a live source in real time use cases. The logical query is converted to a physical plan by the compile-time component of the system which includes the logical to physical translator and optimizer. The physical plan can then be executed either on a single node or in a distributed manner.

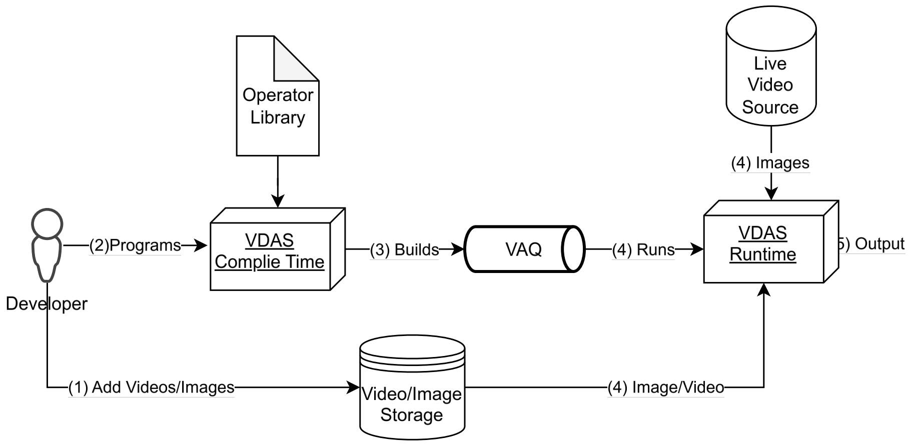


Figure 2.1: General VDAS Design

2.1.1 Comparison with Relational Database Management System (RDBMS)

In Table2.1 we highlight important differences between RDBMS and VDAS. We contrast them on following points:-

- Data Model:- In RDBMS queries operate on a relation also known as table. The output of the query is also a relation which allows composition of queries. In contrast, the 'correct' data model for VDAS is still not agreed upon by the developers and researcher. Different research works have proposed different data models.
- Operators:- Operators in RDBMS is limited to selection, projection, join, union and aggregation. Modern databases allow User Defined Operator (UDO)s but they are not as common. In VDAS, in addition to the traditional operators, domain specific operators are used in almost all the queries. A system cannot come with all possible UDO, hence any usable VDAS must have first class support for UDOs.
- Optimization Target:- In RDBMS the optimizer's goal is to optimize the runtime of the query. However, in VDAS optimizers must also optimize for accuracy of the query. This creates scenarios where a less accurate query is faster and vice versa. In these cases user can specify a target accuracy or latency that the optimizer must satisfy.
- Runtime Environment:- Relational queries run on CPU, whereas VDAS contains ML operators that can run on CPU, GPU or TPU. VDAS can also be distributed across devices, edge and cloud.
- Output:- The output of a RDBMS is deterministic, there is only one correct answer and system is guaranteed to return that. In contrast, VDAS can return erroneous results and the programmer should be aware of that.

	RDBMS	**VDAS**
Data Model	Relation	Patch, VObj, ...
Operators	Limited	UDOs are common
Optimization Target	Query Time	Query Time, Accuracy
Runtime Environment	Homogeneous	Heterogeneous
Output	Deterministic	Stochastic

Table 2.1: Comparison between RDBMS and VDAS

2.2 Visual Operators

We define a visual operator as an operator that takes one or more images/videos as input, does some processing, and produces some result. We have categorized these operators as video encoder/decoder, classical visual operators, and modern ML-based operators.

2.2.1 Video Encoding/Decoding

For efficient storage and transmission, videos are typically encoded and compressed. Typical encoding process involves:-

1. Color Space Conversion:- Video's color space is converted to a more suitable format for compression, often YUV.
2. Spatial Compression:- Reduces redundancy within frames using techniques like Discrete Cosine Transform (DCT).
3. Temporal Compression:- Exploits redundancies between frames using methods like motion estimation and compensation. In this technique only few of the frames called *key frames or I-frames* are encoded completely like a static image. Other frames in the video are represented by the change since the last I-frame. These frames are called *delta frames or P-frames.*

4. Entropy Coding:- Further compresses data using algorithms like Huffman coding.

For decoding the reverse of above steps are performed, (i) Entropy decoding, (ii) Inverse DCT and (iii) Frame Reconstruction using the I-Frames.

The decoding can be performed at the query time which increases overall query time. However, saving individual frames requires more space than the compressed video.

2.2.2 Classical Visual Operators

Low level operators typically focus on pixel-level manipulation and involve basic processing tasks that are crucial for extracting primary features and preparing data for downstream analysis. These operators generally do not involve complex understanding or interpretation of the image content. They are deterministic and involve straightforward mathematical operations and do not require learning. This section provides a non-exhaustive but commonly used classical operators.

Spatial/Temporal Trimming

Spatial trimming involves cropping the frame of an image or a video to a given dimension. More precisely, let I be an image with width=W and height=H, and (x_1, y_1) and (x_2, y_2) be the top-left and bottom-right coordinate of the rectangular region we want to keep, then

- Original Image: $I(x, y)$ for $x \in [0, W]$ and $y \in [0, H]$
- Cropped Image: $I'(x, y)$ for $x \in [x_1, x_2]$ and $y \in [y_1, y_2]$
- And, $I'(x, y) = I(x, y)$, where $x_1 \leq x \leq x_2$ and $y_1 \leq y \leq y_2$

Temporal trimming involves cutting the duration of a video to specified timestamps. More precisely, Suppose the original video has a total duration T and we want to keep a segment from time t_1 to t_2:

- Original Video: $V(t)$ for $t \in [0, T]$
- Trimmed Video: $V'(t)$ for $t \in [t_1, t_2]$
- And, $V'(t) = V(t)$, where $t_1 \leq t \leq t_2$

Convolution

The general expression of a convolution is:

$$g(x, y) = \omega * f(x, y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} \omega(i, j) f(x - i, y - j) \tag{2.1}$$

where $g(x, y)$ is the filtered image, $f(x, y)$ is the original image, and ω is the filter kernel. Every element of the filter kernel is considered by $-a \leq i \leq a$ and $-b \leq j \leq b$. Depending on the element values, a kernel can cause a wide range of effects, such as blurring, sharpening, noise reduction and edge detection.

Affine Transformations

An affine transformation is a linear mapping method that preserves points, straight lines, and planes. In the context of image processing, affine transformations are used to perform

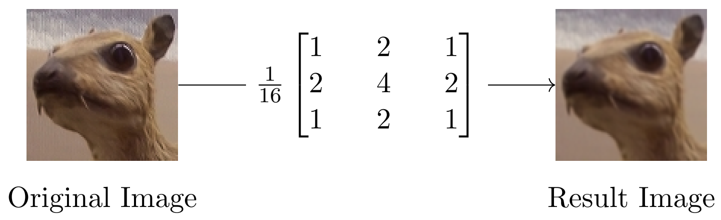

$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$
Original Image
Result Image

Figure 2.2: Illustration of a convolution operation.

a variety of image manipulations, including translation, scaling, rotation, and shearing. Mathematically,

$$\begin{bmatrix}x'\\ y'\\ 1\end{bmatrix} = \begin{bmatrix}a & b & t_x\\ c & d & t_y\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\ y\\ 1\end{bmatrix}$$

where:

- (x, y) are the coordinates of a point in the original image.
- (x', y') are the coordinates of the point after transformation.
- a, b, c, d define the linear transformation (rotation, scaling, and shearing).
- t_x and t_y are the translation components.

Low level feature detection

Edge detection in image processing is a technique used to identify boundaries within an image. These boundaries typically represent significant changes in intensity or color between neighboring pixels. Common algorithms are Canny[6], Sobel[35].

Corner/Interest point detection involves identifying points in an image that are distinct and can be used for further processing, such as image stitching and tracking. Common methods are:- Harris Corner Detection, SIFT and ORB.

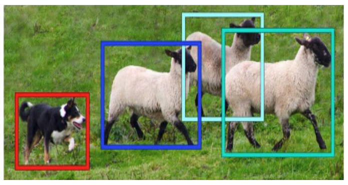
Blob detection identifies blobs in an image. A blob is a region in an image where some characteristics are significantly different from the surrounding area. This could include regions of consistent color, texture, or brightness. Blobs can be thought of as objects or features of interest within the image. Common algorithms are Laplacian of Gaussian (LoG), Difference of Gaussians (DoG) and Determinant of Hessian (DoH).

Shape detection involves identifying and classifying shapes within an image. Using this technique we can distinguish shape based on their geometric properties, such as edges, and structural elements. Common technique includes thresholding, hough transform[15], and template matching.

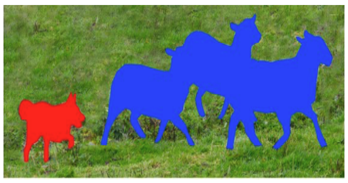
2.2.3 ML-Operators

ML visual operators involve more sophisticated and semantically meaningful processing that goes beyond simple pixel manipulations. They focus on interpreting and understanding the visual content, leveraging ML to infer higher-level concepts and object relationships in the image.

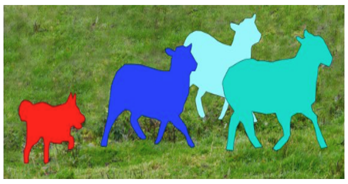
(a) Image Classification



(b) Object Detection

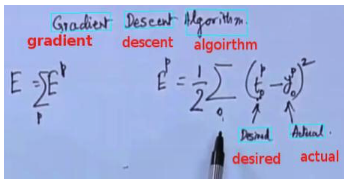


(c) Semantic Segmentation

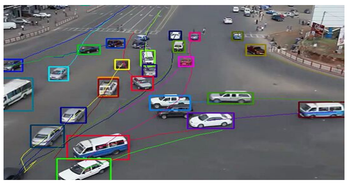


(d) Instance Segmentation

(e) Pose Detection



(f) Optical Character Recognition (OCR)



(g) Object Tracking

Figure 2.3: Visual Analytics Operators

Image Classification

In image classification, the objective is to categorize an input image into one of several predefined classes or categories. More precisely, let X be the space of images, and Y be the set of possible object classes. A function

$$f : X \rightarrow C$$

performs classification task, where $C \in Y$. Common ML models for this task are:- ResNet[41], EfficientNet[36], and BLIP[23].

Object Detection

Object detection involves identifying and locating objects within an image. Unlike image classification, which assigns a single label to an entire image, object detection provides both the classification of objects and their spatial locations through bounding boxes. More precisely, let X be the space of images, and Y be the set of possible object classes, a function

$$f : X \rightarrow (C, B)^n$$

performs object detection task, where n denotes the number of objects present in an image, (c_i, b_i) denotes predicted class label for i_{th} object and $c_i \in Y$, and b_i represents the predicted bounding box coordinates (xmin, ymin, xmax, ymax). Common ML models includes:- YOLO[31], Fast-RCNN[12], SSD[25]

Semantic Segmentation

Semantic Segmentation involves classifying each pixel in an image into a predefined class. Unlike image classification, which assigns a single label to an entire image, and object detection, which localizes objects with bounding boxes, semantic segmentation provides a detailed, pixel-level classification that assigns a class label to every pixel, effectively dividing the image into meaningful segments corresponding to different object categories. More precisely, given an image X of size $W \times H$k and Y be the set of possible object

classes,
a function

$$f : X \to L$$

where $L \in Y^{W \times H}$k can perform semantic segmentation. Common ML models are:- UNet[34], and DeepLab[9].

Instance Segmentation

Instance segmentation involves identifying and segmenting each object instance in an image. Unlike semantic segmentation it not only distinguishes between different object classes but also separates individual instances of the same class. Each instance is segmented out and assigned a unique identifier. Given an image X of size $H \times W$, and Y be the set of objects present in the image.
A function

$$f : X \to L$$

where $L \in Y^{W \times H}$, can perform instance segmentation task. Common ML models includes:- MaskFRCNN[14].

Pose Detection

Pose detection involves identifying landmarks of the human body(or any other articulated objects) in an image. The goal is to determine the precise position and orientation of these key points to understand the pose and movement of the body. Landmarks are the specific locations on the body such as joints(e.g., elbows, knees) in case of human body. Let X be the image and Y be the set of landmarks of an object.
A function

$$f : X \to (A, B, K)^n$$

where n is number of landmarks detected in the image, $k_i \in Y$, and (a_i, b_i) is a pixel in the image. Common technique includes:- OpenPose[7], PoseNet[21], and DeepPose[38].

Object Tracking

Object tracking involves following the movement of objects within a video or across a sequence of images. The goal is to determine the position and trajectory of an object over time, ideally even in presence of occlusion, change in appearance or background clutter. Tracking is of two types:-

- In **Single-Object Tracking (SOT)** a single object is tracked throughout the video, the object of interest is specified by the user
- In **Multi-Object Tracking (MOT)** multiple objects are tracked, depending on the objects being detected by the object detector. This requires us to keep state of every object currently being tracked, as well as add new objects in the state when they are first discovered and remove objects from the state when they are no longer visible.

Re-identification (ReID)

ReID involves recognizing the same entity across different camera views or times. Difference between object tracking and ReID is that the later might need to store larger state than object tracking since the same objects might appear across a larger timeframe.

Optical Character Recognition (OCR)

OCR involves converting different types of documents, such as scanned paper documents into editable and searchable data. Primary function of OCR is to recognize text characters in the image. More precisely, given an image X an Y be a set of characters.
A function

$$f : X \to 2^Y$$

where 2^Y is the powerset of Y. Common technique include: Easyocr[24], Tesseract[20], and large multimodal models like Gemini[37] can also be used to extract text from images.

Image Embeddings Extraction

An image embedding is a vector that encodes the semantics of contents of the image. This can be used to compare similarity of two images which is useful when doing image search. These embedding become proxy for the image and cosine distance (or other distance metrics) between them becomes proxy for similarity. Embeddings are usually extracted from a Convolutional Neural Network (CNN)'s penultimate layer. In recent years multimodal (text and images) models such as CLIP[30] are also being used for image embedding extraction. They are trained with a large dataset of pairs of text and image, hence they are able to understand relationship between images and text, allowing image search with a text query.

2.3 Visual Analytics Query

VAQ is logical representation of a visual analytics task. It typically includes multiple visual and non-visual operators stitched together in order to reach desirable output. Fig. 2.4 shows an example of a VAQ that outputs the timestamp of the frames where a car is found with certain license plate. ML operators are shown in gray boxes.

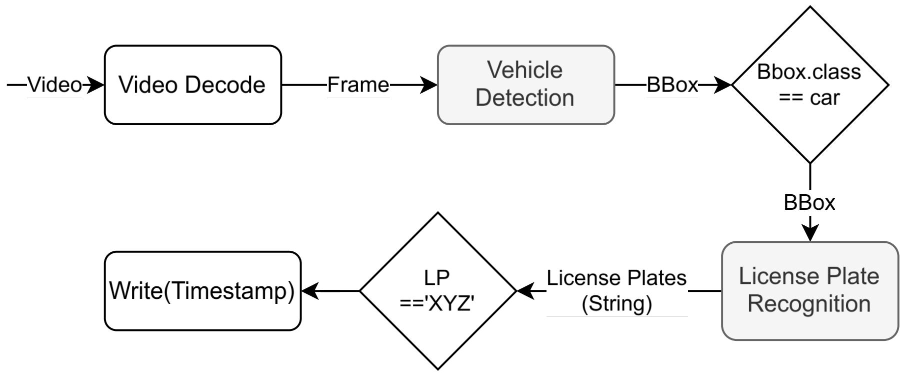


Figure 2.4: Example of a VAQ

2.4 Common Use cases

2.4.1 Traffic Analysis

Increased traffic in urban environments can lead to more accidents and congestion if proper traffic management strategies are not implemented. Intelligent video analysis solutions can be highly effective.

These solutions can be used for following tasks:-

- Automatic traffic violation detection: These systems can identify and record traffic violations such as speeding, running red lights, and illegal turns, thereby improving road safety and enforcing traffic laws.

- Amber alert: Video analysis can assist in issuing Amber Alerts by identifying vehicles or individuals that match descriptions in cases of child abduction or other emergencies.
- Automatic adjustment of traffic light systems: Real-time traffic data can be used to optimize traffic light timings, reducing congestion and improving the flow of vehicles.
- Traffic Statistics: Video systems can collect data on traffic volume, vehicle types, and travel times, which can be used for planning and improving transportation infrastructure.
- Smart parking system: These systems can detect available parking spaces and guide drivers to them, reducing time spent searching for parking and improving overall traffic flow.

2.4.2 Healthcare

- At-home monitoring for elderly: Video systems can monitor patients at home for signs of distress or abnormal behavior, providing valuable data for early intervention and reducing the need for hospital visits.
- Mental healthcare: By analyzing facial expressions and body posture, video systems can help in the assessment of mental health conditions such as depression and anxiety, enabling timely support and treatment.

2.4.3 Retail

In the retail sector, video analysis provides insights into customer behavior and store operations, helping to enhance the shopping experience and improve store performance.

- Determine customer's characteristics: Video systems can analyze customer demographics, allowing retailers to tailor marketing strategies and improve customer service.
- Analyze walking patterns and duration of visit: Understanding how customers move through a store can help optimize store layout and product placement to increase sales. Tracking the time customers spend in different areas of a store also provides insights into their interests and engagement, informing decisions on merchandising and product promotion.

2.4.4 Sports Analytics

- Player tracking and performance analysis: We can track player's movement to analyze their position, speed and stamina. Video analytics can also calculate performance metric such as distance covered.
- Ball Tracking: Advanced video systems can track the ball's movement during games, providing valuable data for performance analysis, game strategy, and enhancing the viewing experience for fans.
- Fair Play: VDAS can provide assistance to referees by doing real-time foul detection.

2.5 Datasets

Human labelled datasets are required for training and evaluation of accuracy of a ML models. There is a huge amount of dataset available for computer vision task, so to keep the list concise, we only list the datasets that are frequently used in related works in Table 2.2.

Name	**Description**	**Tasks**
ImageNet	Well known object classification dataset with 1000 different classes	Object classification
COCO	Well known object detection dataset with 80 different classes	Object detection
Pascal VOC	Well known object detection dataset with 20 different classes	Object detection
Objects365	Well known object detection dataset with 365 categories and 2M images	Object detection
VisDrone	Contains 288 video clips captured by drone.	Object Detection & Tracking
BDD	Consists of 1100 hours of video from dashboard cameras on motor vehicle	Car, pedestrian and sign detection
UAV	Consists of 2 hours of video from UAV hovering above a traffic junction	Car, pedestrian and sign detection
taipei, night-street, rialto, grand-canal, amsterdam, archie, coral-reef	Commonly used dataset in related works	Vehicle detection & tracking
CelebA	large-scale face attributes dataset with more than 200K celebrity images, each with 40 attribute annotation	Facial Attribute recognition
UCF101	video collected from YouTube, having 101 action categories	Action recognition
Cable TV News Dataset	The dataset includes near recordings of various TV Channels with caption and person detection	Face Recognition
CityFlow-NL	Includes multiple dataset for smart city tasks.	Tracking and Retrieval using Natural Language

Table 2.2: Visual Datasets

3 Related Work

In this section we will describe the related existing systems in the area of video analytics. This will help us understand the landscape of current video analysis. These works are listed in ascending order of their publication year.

3.1 Video Analytics Systems

3.1.1 Optasia [26]

Optasia was one of the earliest attempt at scaling video analytics. They build on top of Microsoft's SCOPE[8] system, and uses SCOPE's interface and runtime. They recognized that:-

- Different queries running simultaneously may have common intermediate output which can be reused.
- Same query might have operators with common intermediate operation which can be reused. Fig3.1 shows 3 query queries: amber alert, traffic violation and amber alert + traffic violation. As we can see in Fig3.1(c) common operators are deduplicated to minimize the query cost.

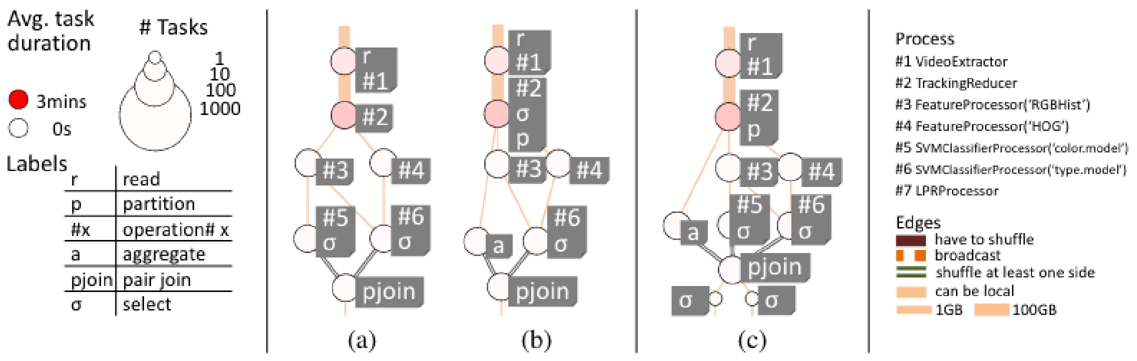


Figure 3.1: Example Optasia Query

3.1.2 NoScope [19]

Researchers observed that the computational cost associated with applying object detectors to every frame of a video is too high. This process can be particularly resource-intensive and time-consuming. To address this, researchers proposed two innovative ideas:

- Difference Detector

  The first idea is to develop of a difference detector. This detector is designed to identify significant changes between consecutive frames in a video. The core idea is to only invoke the more expensive object detector when a notable change is detected by the difference detector. This approach significantly reduces the number of frames that require full analysis, thereby saving computational resources. The difference detector is considerably faster and less resource-intensive compared to the full object detector.

- Specialized Models

  The second idea is the concept of specialized models. Traditional general-purpose object detectors are typically large and capable of identifying a wide range of objects, such as buses and apples. However, in many real-world scenarios, the requirement is often to detect a much narrower set of objects. For instance, in a traffic monitoring system, the primary interest might be in detecting only vehicles, not fruits or other unrelated objects.

  To optimize for such specific use cases, the researchers proposed training smaller, specialized models tailored to the particular objects of interest. These models can be trained using labels generated by the general-purpose detector, focusing solely on the relevant subset of objects. This leads to models that are more efficient and faster while maintaining accuracy for the targeted detection tasks. These smaller models are trained on the labels produced by the larger model, during the video ingestion time.

The second idea is very important and is refined further by later works. In Figure3.2 we see comparison between traditional Deep Neural Network (DNN) inference and optimized inference, at every frame only difference detector is run, which is the fastest operation, if a difference is detected then only a lightweight proxy model is run, if proxy model indicates that bus is present then only the heavier model is run.

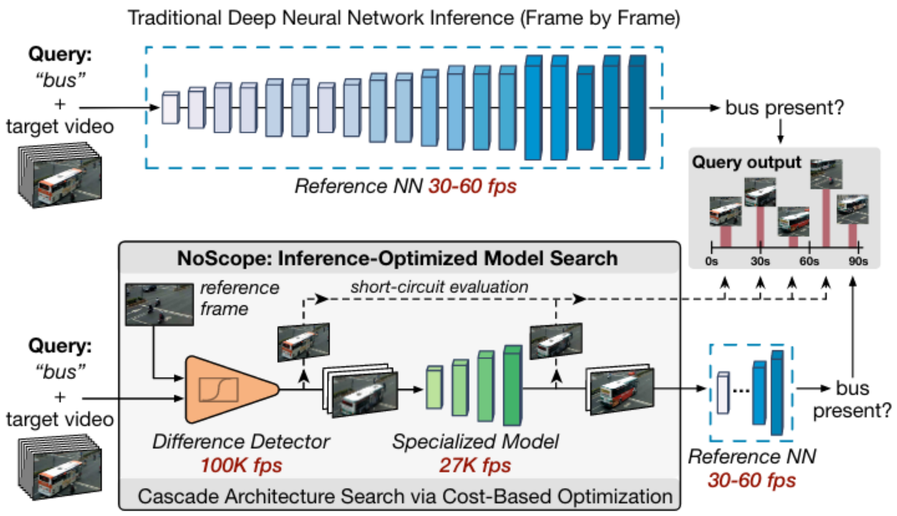


Figure 3.2: NoScope Architecture

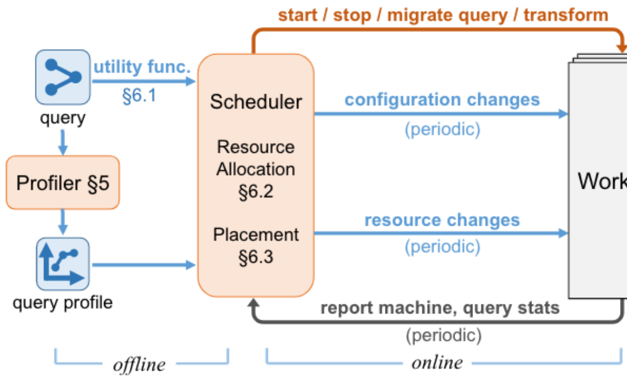
3.1.3 VideoStorm [42]

Researchers recognize that a fair scheduler allocates equal resources among many queries. This may not be desirable, since different queries might have different resource requirement. Hence, they build a scheduler for visual queries with following properties:-

- **Allocate more resources to queries whose quality will improve more**. For example, say adding additional resource improves quality of a query from 0.7 to 0.9 and same addition can improve the quality from 0.3 to 0.7 for some other query. An intelligent scheduler should allocate the resource to the latter query since its quality will improve more. To get this information, profiling is done in an offline phase.

- **Allow queries with built-up lag to catch up** (when resources are freed up). A query with under allocation of resources will lag behind, so to when the resource is finally available the scheduler should allocate the resources to the lagged behind query. This is done by monitoring running queries in the online phase.
- **Adjust query configuration based on allocated resource**. The quality and lag requirements of a query are encoded in a utility function. The performance goal across all queries in a cluster is specified either as maximizing the minimum utility or the sum of utilities.

A license plate recognition query is shown in Fig3.3b



(a) VideoStorm Scheduler

```
  "name": "LicensePlate",
  "transforms": [
    {"id": "0",
     "class_name": "Decoder",
     "parameters": {
       "CameraIP": "134.53.8.8",
       "CameraPort": 8100,
       "@OutputResolution": "720P",
       "@SamplingRate": 0.75 }
    },
    {"id": "1",
     "input_transform_id": "0",
     "class_name": "OpenALPR",
     "parameters": {
       "@MinSize": 100,
       "@MaxSize": 1000,
       "@Step": 10 }
    } ]
```

(b) VideoStorm Query Example. Parameters prefixed with @ are the configuration knobs.

Figure 3.3: VideoStorm Overview

3.1.4 Focus [16]

Researchers criticize NoScope because there can be ‘any number of object classes’ that could be queried later and running even lightweight binary classifiers for many classes can be expensive. At ingest-time, Focus classifies the detected objects using a cheap CNN, clusters similar objects, and indexes each cluster centroid using the top-K most confident classification results, where K is auto-selected based on the user-specified precision, recall, and cost/latency trade-off point. At query-time, Focus looks up the ingest index for cluster centroids that match the class X requested by the user and classifies them using the GT-CNN Finally, Focus returns all objects from the clusters that are classified as class X to the user. The idea of indexing at ingest time is also very important and refined in the future works.

In fig3.4 we see that at ingest time top-k index is created, and at query time instead of running a heavy CNN on all the frames, we run it only on the frames whose centroids are classified as class X.

3.1.5 Chameleon [17]

Researchers claim that figuring out a suitable query configuration once before the execution is not enough. The best configuration now may not be the best after a while. For example, we may use a low frame-rate when cars are moving slowly without impacting the accuracy of the vehicle count. In contrast, using the same configuration when traffic is moving fast will hurt the accuracy. They realized that naively doing periodic profiling different configurations is not optimal as there can be exponentially as many. They exploit the following three properties to perform the profiling more efficiently:-

- **Persistent characteristics over time**:- Good configurations tend to consistently produce good performance. So no need to consider all configurations. Consider only top-k 'good' configurations.
- **Cross-camera similarity**:- Cameras that observe similar scene will have similar optimal configuration. So we can share top-k configurations between them.
- **Independence of configuration knobs**:- Researchers observed that typically individual knobs have independent impact on accuracy. So exploration space reduces to linear from exponential.

Following technique is used to exploit the above-mentioned properties:-

- A "leader" video is profiled at the start of a "profiling window" and a set of good (top-k) configurations is found.
- This set is shared among "follower" videos who are similar to the leader.
- Both the leader and followers then restrict their search to this top-k set when choosing configurations over time, until the start of the next profiling window (when a new top-k set will be obtained from the leader).

Figure3.5 shows how the above technique works in chameleon. The horizontal lines represent video feeds generated by three cameras (one leader, two followers). The solid vertical lines separate the profiling windows. The blue circle represents profiling, and dark blue circle is full profiling which is only done on leader. The red arrow shows the propagation of the top-k configs both temporally (within a camera) and spatially (to other cameras).

3.1.6 DeepLens [22]

Researchers try to create the 'narrow waist' for visual analytics. In that attempt they build an end-to-end system, while charting future works. The abstraction they used is a *Patch*, which contains reference to the image that created it, main data of the patch, and

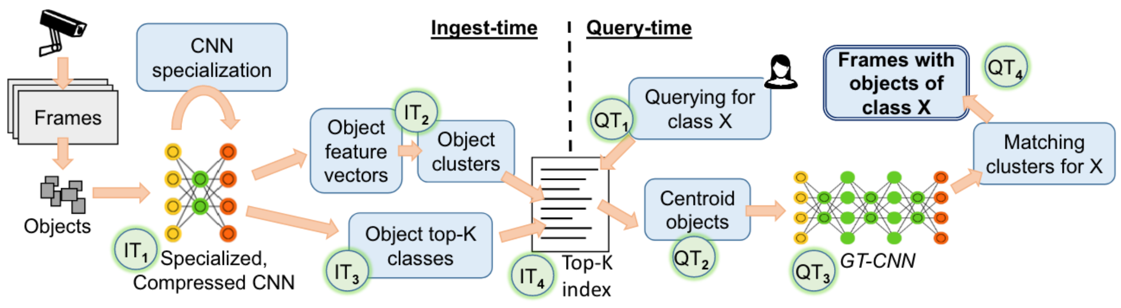


Figure 3.4: Focus Architecture

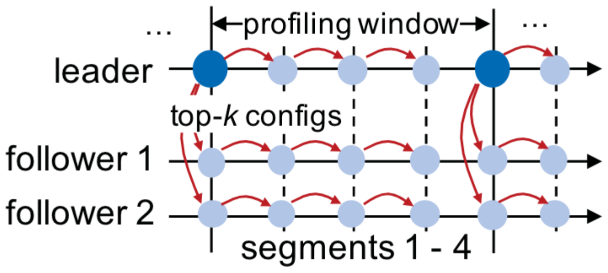


Figure 3.5: Chameleon Technique

a key-value metadata. Operators work on an iterator of patches and outputs an iterator of patches. The system's overview in figure3.6 is as follows:-

- **Storage Layer** can store images as well as derived data such as bounding boxes. Queries for image fetching also support filtering on channel, timestamp. The videos can be stored as individual frames requiring no decoding, as well as an encoded file. Multiple types of index are supported for efficient retrieval of derived data.
- **Visual Etract-Transform-Load (ETL)**:- At image load time *patch generators* are run that turns the image into a set of patches. This includes whole-image patches, object detection/segmentation and OCR. For future work automatic query synthesis is suggested.
- **Query Processing**:- Select and joins can be performed on the patches. If indexes are available index joins can be performed else nested loop joins will be performed. For future work, a query optimizer is suggested that optimizes accuracy as well GPU/CPU. Researchers recognize following subtleties in QO:-
  - **Balancing CPU vs GPU**: Researchers note that for smaller queries certain tasks on CPU with AVX can outperform the GPU, however GPU scales better, so selecting placement of operator is also very crucial.
  - **Accuracy is not additive or multiplicative**:- Unlike relation databases visual query results are approximate in nature. Cascade of approximate operators can have correlation that are hard to reason with. For example:- Consider a query *Count all distinct pedestrian in a video.* Two approaches are possible to answer this query:- Filter all pedestrians first, then do a matching of all pedestrians to deduplicate. Or match first then filter the pedestrians. The second approach will have higher latency since we didn't filter anything before matching. However, the second approach will have higher recall (In second case if either of two patch is of pedestrian then we consider it a match).

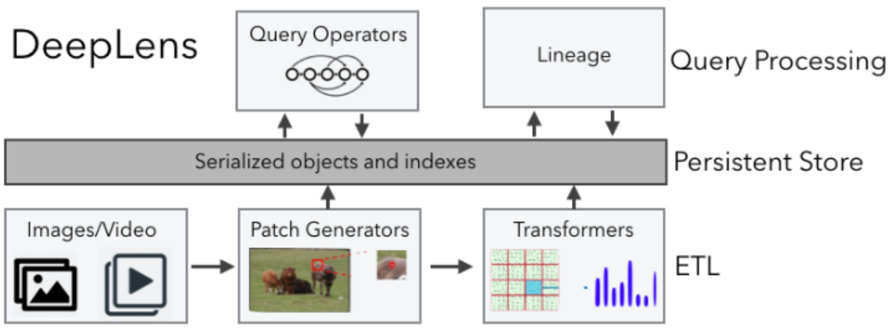


Figure 3.6: DeepLens Architecture

3.1.7 Tahoma [1]

Researchers builds on top of NoScope while criticizing it for concentrating only on computation and ignore the inevitable data-handling costs, such as loading and transformation.

As an example, consider an optimizer choosing between two image classification models ($M1$ and $M2$). $M1$ accepts a 3-channel, full-color, 224×224 image as input while $M2$ accepts a 1-channel, grayscale, 224×224 image. $M1$ has fewer convolutional layers and, despite the larger input, requires fewer tensor operations than $M2$, so its inference is faster. $M2$ uses less rich data than $M1$, but is nonetheless able to obtain comparable accuracy because of its additional convolutional layers. An optimizer that considers only the inference speed would choose $M1$. However, a data system using $M2$ might be faster, as its inputs load in one-third the time of those of $M1$.

Following tasks are performed at ingestion time:-

- train many specialize candidate CNN based binary-classifiers by varying not only CNN hyperparameters but also the input representation. Using this candidate models many classifier cascades are constructed.
- An optimization method evaluates the cascades' accuracy using held-out data.
- Final Pareto-optimal cascades that satisfy the user's application-specific speed and accuracy is identified.

Classification using a cascade is done as following:- The models in the cascade are run in series: image I_i is classified by M_1, and if the output between two given decision thresholds, p_{low} and p_{high}, it is uncertain. So I_i is then classified by M_2. Otherwise, the cascade is stopped and M_1's is accepted. This continues to the final classifier M_n whose output is always accepted. In figure3.7 we see classifier cascade this in action.

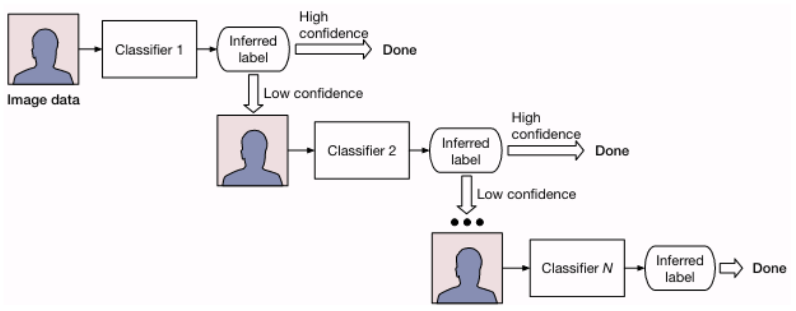


Figure 3.7: Multilevel classifier cascade

3.1.8 BlazeIt [18]

Researchers point out that previous systems (NoScope and Tahoma) do not support aggregation and limit queries. They extend SQL for visual queries (fig3.8). Fundamental idea is to enable aggregation and limit query by training a proxy model that given an image quickly tells how many object of interest is present in the image without running a detector.

- **Supporting fast aggregates**:- Researchers train a custom tiny Resnet model to directly answer the aggregate query, i.e how many objects in the image. Object detection models would be slow here since they also give out bounding boxes and labels for those objects, whereas this model only count occurrences. The trained proxy model is applied on all the video frames. To generate labeled data for training the tiny Resnet model, mask r-cnn oracle model is applied on a small subset of frames to derive the answer for the aggregate query (e.g., number of cars).
- **Supporting fast limit query**:- Limit queries are helpful for rare events. We could perform object detection on every frame to search for the events, but if the events are infrequent, random sampling or sequential scans can be very slow. Key intuition is to bias the search towards region of the video that likely contain the event. For this BlazeIt uses the same model trained for aggregates. The frames are sorted by count and confidence given by the specialized NN. The full object detector is then applied until the requested number of objects are found.

3.1.9 Panorama [44]

Researchers tackle the problem of 'unbounded vocabulary'. The object recognition CNNs only handle a finite vocabulary of known targets. This is the direct consequence of their training dataset. For example, Pascal VOC has only 20 classes. So models trained on it can only tell apart these 20 classes. But a real-world query might include a class that the model is not trained on. In such cases, the researchers provide a solution with three main components:-

- **Unified CNN architecture PanoramaNet**:-
  - *Multitask learning*:- Same model does object detection, recognition as well as verification.
  - *Out of vocabulary recognition*:- is done by doing KNN search using the embedding of known objects and query object.
  - *Enabling cost and accuracy tradeoff*:- The lowest layers of the CNNs act as a shared feature extractor for all blocks. All output layers have the same semantics, but they offer different accuracy-runtime tradeoffs. By short-circuiting at an earlier block, the inference cost goes down.
- **Automated offline training**:- A user-given reference model is run on a portion of video to create training data for the training of PanoramaNet.
- **Online inference with short-circuiting**:- The network is made up of several blocks. Inclusion/Exclusion of blocks create an effect of many CNNs chained together, that allows short-circuiting as in Tahoma. A score is computed for a given query at each block and compared it against a pre-computed 'cascade interval'. If the query's score at a block falls in it cascade interval then the subsequent blocks is invoked otherwise short-circuit happens.

```
SELECT FCOUNT(*)             SELECT timestamp
FROM taipei                  FROM taipei
WHERE class = 'car'          GROUP BY timestamp
ERROR WITHIN 0.1             HAVING SUM(class='bus')>=1
AT CONFIDENCE 95%               AND SUM(class='car')>=5
                             LIMIT 10 GAP 300
```

Figure 3.8: Aggregate and Limit Query in Blazeit

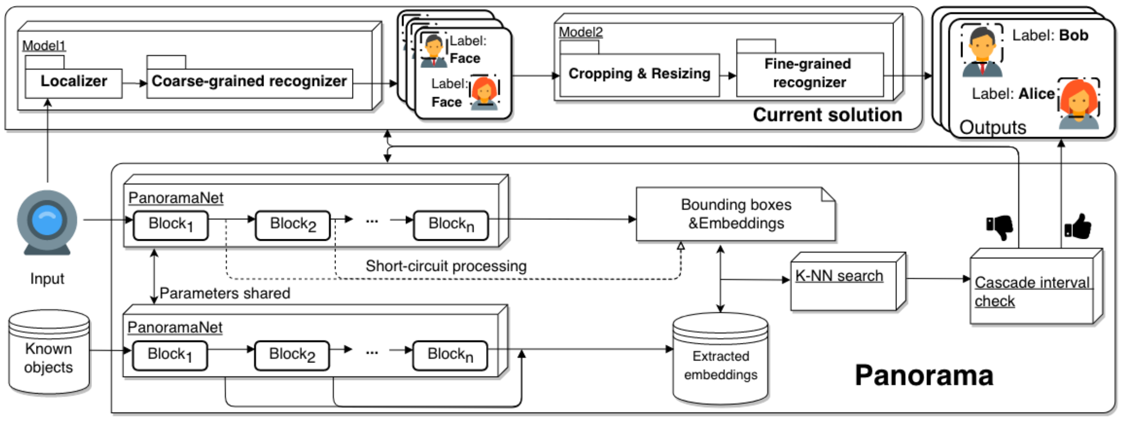


Figure 3.9: Panorama Architecture

3.1.10 Miris [4]

This work focuses on object tracking which involves multiple frames of a video, as opposed to previous works which only focused on per-frame predicates. Researchers claim that proxy model idea does not work here because, when tracking an object, that object will appear on every frame for a while, and the proxy model will not filter any of the frames and as a result expensive object detectors will be applied on every frame.

Researchers propose that by controlling the frame rate at which the video is sampled, they can accelerate query execution. If objects can be tracked accurately at say 1 fps instead of 25 fps then the cost can be substantially reduced. Following technique is used for the optimization.

- Object detection and tracking is performed on the video at reduced frame rate to obtain object tracks. Due to the low sampling frame rate, some tracks will contain uncertainty where the tracking algorithm is not confident that it has correctly associated a sequence of detections.
- To increase confidence additional frames are sampled in the uncertain tracks.
- However, if we are certain that the uncertain tracks do not satisfy the predicates anyway we do not need to do additional sampling. This is the key idea.

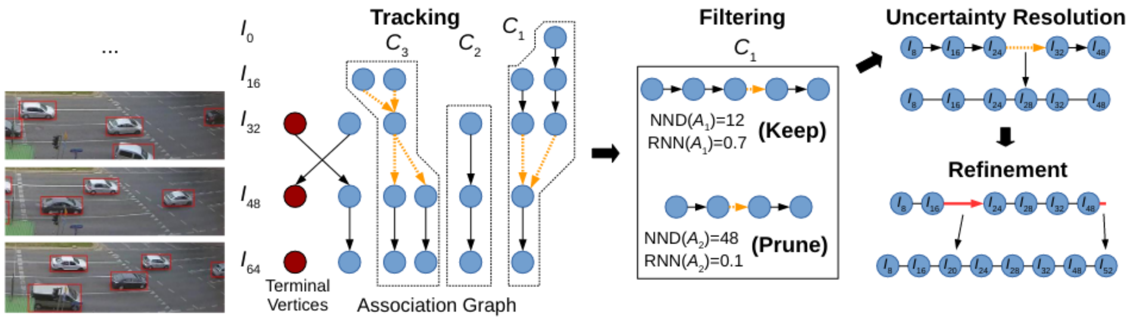


Figure 3.10: Orange dashed edges result from uncertain matching decisions, and yield multiple nondeterministic tracks, which are resolved during uncertainty resolution. Refinement identifies segments of tracks that require additional detections for accurate predicate evaluation.

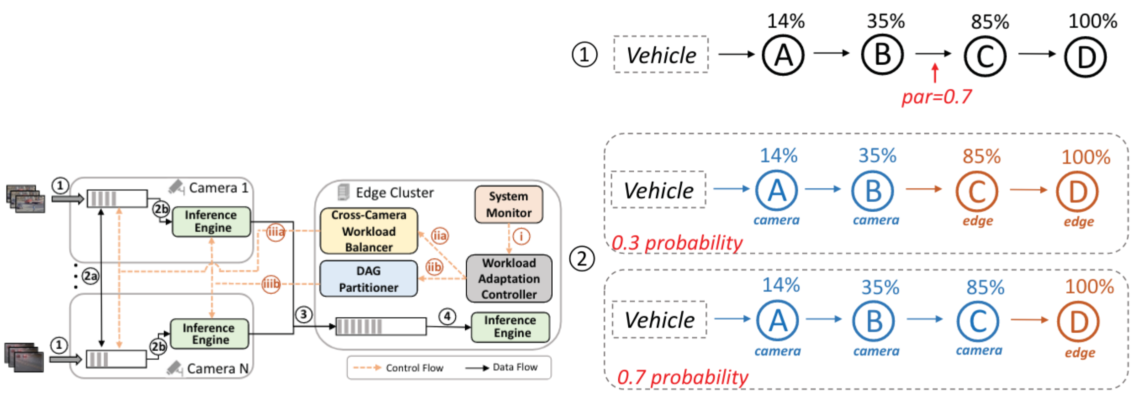
3.1.11 DiStream

Researchers distribute the Deep Learning (DL) based workload between smart camera and a centralized cluster.

Typical data flow in the system:-

- The frame to be processed is appended in a local queue inside the camera.
- The frame can be processed either processed locally or offloaded to other camera decided by *cross-camera workload balancer*.
- Only some part of processing is done on camera rest is offloaded to cluster. The partition is *stochastic* determined by a DAG *partitioner*. Take the 'Vehicle' path in Figure 3.11b as an illustrative example. Assume we need to achieve 0.7 workload partitioning (i.e., 70% workload allocated to the camera and 30% workload allocated to the edge cluster) determined by the workload adaptation controller. The stochastic DAG partitioning scheme first identifies two vertexes (vertex B, vertex C) along the 'Vehicle' path such that 0.7 falls between their normalized accumulated inference

cost (Figure 3.11b (1)). Given the normalized accumulated inference costs of these two vertexes (0.35 for vertex B, 0.85 for vertex C), the stochastic DAG partitioning scheme then determines the probabilities for partitioning the 'Vehicle' path at vertex B (P_B) and vertex C (P_C) respectively as: $P_B = (0.7 - Cost_C)/(Cost_B - Cost_C)$, and $P_C = 1 - P_B$



(a) Architecture (b) Stochastic Partitioning

Figure 3.11: Distream

3.1.12 JellyBean [39]

Researcher build a system that given resources availability, required latency and target accuracy optimizes the serving cost by selecting appropriate ML models and worker assignment (edge-cloud).

Overall schematics of the system is as follows:-

- **Input** to the system is a logical plan graph G, for each ML operator a list of candidate models with their accuracy profile, required accuracy, and infrastructure specification that contains set of each type of workers each tier has, cost of each type of worker, and the communication cost between different tiers.
- **Accuracy Profiling** of a model is done by varying parent model and seeing the resulting accuracy. For e.g. consider a model with two inputs this model can have the following accuracy profile: (60%,50%) $\rightarrow$ 55%, (50%,60%) $\rightarrow$ 60%, and (70%,90%) $\rightarrow$ 65%.
- **Model Selection**: The graph is traversed in reverse topological order, and assign the model for each node. Each candidate will have an accuracy requirement from the parent operators, these requirements are propagated to the parents and parents are chosen accordingly. After this each node might have multiple candidates, candidate with the lowest cost chosen.
- **Worker Assignment**: The graph can be partitioned once between edge and cloud, and appropriate partition needs to be chosen. Researchers provide a greedy algorithm that assigns workers to each partition.

3.1.13 Otif [3]

This work focuses on object tracking queries. In this system all the tracks are extracted from the video during ingestion time. At query time, there is no need to perform video analytics, metadata extracted during the ingestion time is enough to answer the query. Hence, making it much faster than any system that does query time processing.

Following optimizations are also performed during ingestion to make the preprocessing fast as well:-

- **Segmentation proxy model** identifies the regions of frames that contain no objects, and skip object detection processing on these regions.
- **Recurrent reduced-rate tracker**:- Instead of using a Graph Neural Network (GNN) as in Miris, a Recurrent Neural Network (RNN) architecture is used which faster, and is more accurate at lower sampling rate.
- **Joint parameter tuning**. There are 5 parameters that can affect the accuracy and latency of the query. They are:- (i) Resolution and (ii) Threshold in segmentation proxy model, (iii) Resolution and (iv) Confidence threshold in object detector, and (v) sampling gap in the tracker. The multiple Pareto-optimal configuration of these 5 parameters are calculated by profiling on a validation set. They will have different accuracy speed tradeoff, user can select one of the configuration.

3.1.14 Viva [32]

Researchers develop an end-to-end interactive video analytics platform, which includes a (i) video file manager, (ii) execution engine, (iii) query optimizer and (iv) embedding cache.

Notable contributions:-

- **Relational hints**, that allows users to express relationships between columns that are difficult to automatically infer, for e.g., mentions of a person in a transcript can

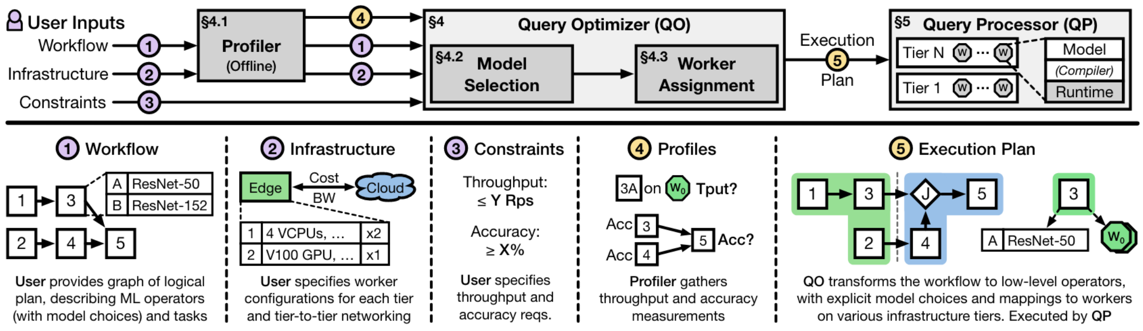
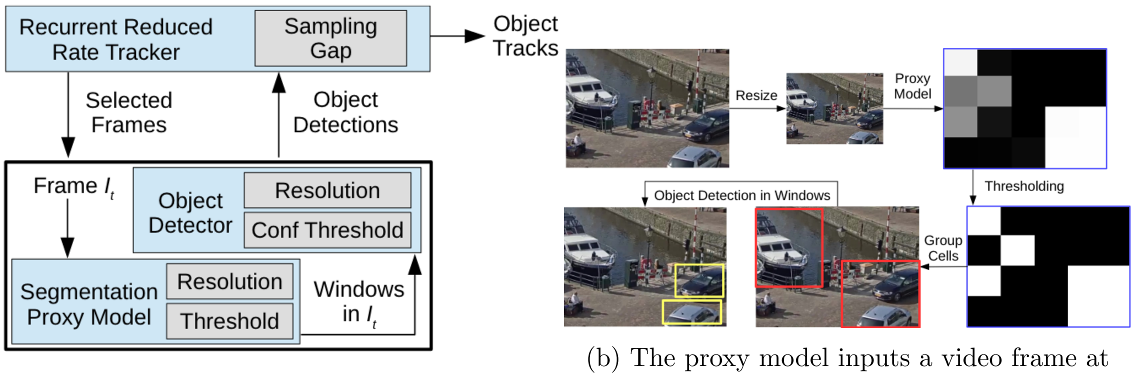

User Inputs
Workflow
Infrastructure
Constraints
§4.1 Profiler (Offline)
§4 Query Optimizer (QO)
§4.2 Model Selection
§4.3 Worker Assignment
Execution Plan
§5 Query Processor (QP)
Tier N
Tier 1
Model
(Compiler)
Runtime
1 Workflow
ResNet-50
ResNet-152
User provides graph of logical plan, describing ML operators (with model choices) and tasks
2 Infrastructure
Edge
Cost
BW
Cloud
4 VCPUs, ... x2
V100 GPU, ... x1
User specifies worker configurations for each tier and tier-to-tier networking
3 Constraints
Throughput: ≤ Y Rps
Accuracy: ≥ X%
User specifies throughput and accuracy reqs.
4 Profiles
3A on W0 Tput?
Acc
Acc?
Profiler gathers throughput and accuracy measurements
5 Execution Plan
QO transforms the workflow to low-level operators, with explicit model choices and mappings to workers on various infrastructure tiers. Executed by QP

Figure 3.12: JellyBean Architecture



(a) The tracker selects which frames to process. To detect objects in each sampled frame, the segmentation proxy model determines which windows of the frame may contain objects, and the detector runs in those windows.

(b) The proxy model inputs a video frame at low resolution, and scores each cell in the frame with the likelihood that the cells intersects a detection. Positive cells after thresholding are grouped into rectangular windows, and the object detector is applied only in those windows.

Figure 3.13: Otif

be used as a proxy for the person appearing in the video.

- **Multiple Supported Queries**:- Prior works focus on single query type, whereas VIVA supports different types of queries which includes:- *selection*, *aggregation*, *limit*, *similarity* and *joins*.

3.1.15 Seiden [2]

Researchers found that difference between speed of oracle and proxy model is not significant enough for the trade-off in accuracy. Instead, they come up with an alternative plan to generate more accurate proxy labels faster than proxy models. Following is the process:-

- Index Construction (At ingest time):- Label is extracted from each I-Frame video of the video using the oracle model. I-frames can be decoded independently, and I-frames are created by the codec when there is drastic change in the scene, and they tend to be evenly distributed.

- Multi Armed Bandit (MAB) sampler (At query time):- Using MAB sampler more frames between the I-frames are sampled and labelled, for the query to get better estimate.

- Label propagation (At query time):- Proxy labels of the unlabeled frame is propagated from the closest orcale labeled frames.

Using the proxy labels, we can answer the aggregate queries fast, without having to compute labels for all the frames.

3.1.16 Zelda [33]

This work leverages Vision Language Model (VLM) that allows following advantage:-

- Natural language interface:- Other systems use SQL like interface that has limited predicate expressivity compared to natural language.

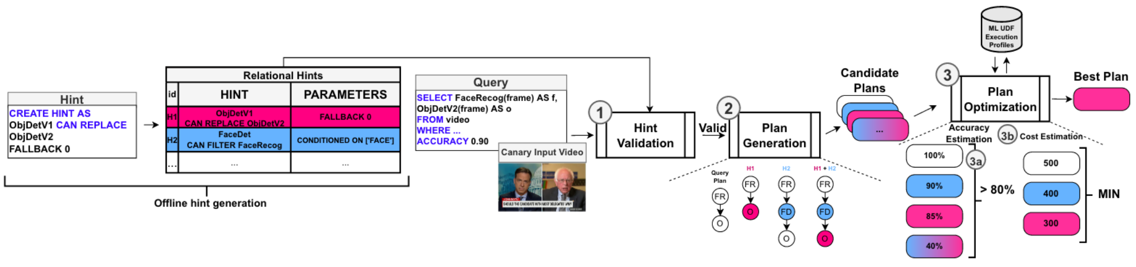


Figure 3.14: Viva Query Optimization, using Relational Hints

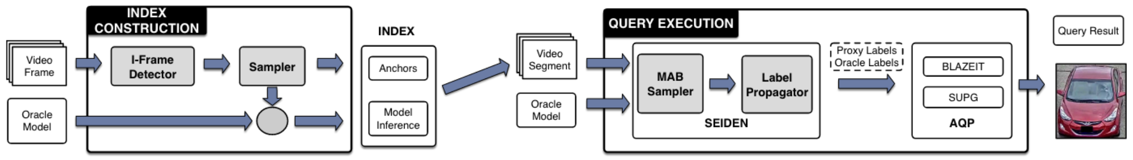


Figure 3.15: During index construction oracle is run on the I-frames to construct the index. During execution, SEIDEN selects additional frames and run the oracle on the selected frames. Finally, the oracle labels are used as the proxy labels for other frames in the video.

- Single general purpose model:- Other systems can only use predicates whose corresponding models are available. By using a single large model we can do arbitrary querying.
- Reduced system complexity:- Using a single model reduces model management overhead.

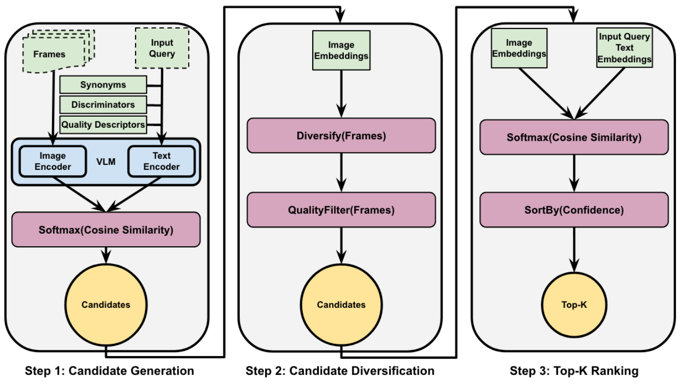


Figure 3.16: Zelda Architecture

To answer a query following steps are taken:-

- Both image frames and input query is passed to the VLM, and using cosine similarity few candidate frames are chosen.
- Apply 'diversify' filters that remove almost similar frames and, quality filter that removes blurry, grainy frames.
- Finally, sort by similarity and show top-K.

3.1.17 VQPy [40]

Researchers develop an object-oriented unified interface for visual queries. Researchers argue that SQL based interface is not ideal for handling object based video queries. Major contribution includes:-

- A video-object-oriented data model: Python is used by users to create 'Visual Object' and 'Visual Queries'. VObj is any object of interest that needs to be extracted from the video, e.g. cars, person. They can have stateless and stateful properties. Visual queries are of three types:-
  - *Spatial Query*:- To express constraint between two objects in the same frame, e.g. distance between person and car in a frame.
  - *Duration Query*:- To express some event occurring over time, e.g. a person loitering from more than 20 mins.
  - *Temporal Query*:- To express some related event that occurs across different frames, e.g. a car hitting a person and speeding away.
- An extensible optimization framework:- The optimization framework can be used to express the optimizations developed in previous works, such as filtering using proxy models.

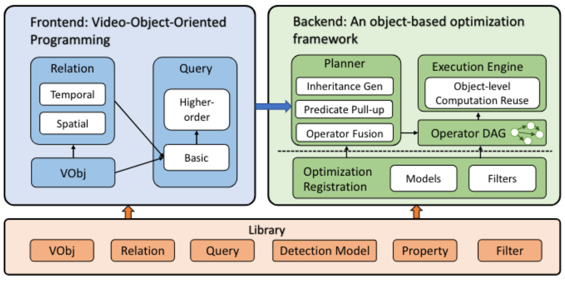
3.1.18 Ipa [11]

Researchers perform multi objective (accuracy and latency) query optimization. The user can specify the desired latency, and system will come up with an optimal plan that matches the latency requirement while providing the best accuracy possible with the required latency. Main contribution includes:-

- Query accuracy estimation:- Since exact accuracy of a query is not required and any estimate that preserves the order is sufficient, researchers simply multiply the individual accuracy of the models involved in the query, to arrive at query accuracy estimate.
- Profiler:- profiles the latency of different models for different batch sizes while varying Requests Per Second (RPS). They find that increasing the memory after a certain value (depending on model size and batch size), does not affect latency. So they focus only on CPU cores.
- Optimization Formulation:- Researchers use Integer Programming to formulate the optimization target, and Gurobi solver to solve the IP problem.

3.1.19 Vulcan [43]

Researchers are trying to solve following 4 problems:



(a) VqPy Architecture

```
from vqpy.lib.query import CollisionQuery, SpeedQuery
from vqpy.lib.query import SequentialQuery

query HitAndRun(vqpy.Query):
  def __init__():
    self.car = Car()
    self.person = Person()
    car_hit_person = CollisionQuery(
        subqueries = [self.car, self.person]
        dist_threshold = 0.1)
    car_run_away = SpeedQuery(car, velocity_threshold)
    self.sequential = SequentialQuery(
        subqueries=[car_hit_person, car_run_away],
        time_window="10s",
        id_match=(car_hit_person.car, car_run_away.car)
    )

  def video_constraint():
    return self.sequential

  def video_output():
    return self.car.license_plate
```

(b) Hit and Run query in VqPy. CollisionQuery is a spatial query, SpeedQquery is duration query, and SequentialQuery is a temporal query.

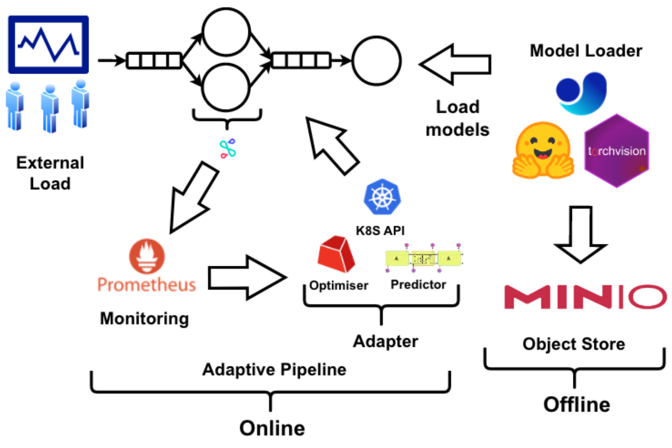


Figure 3.18: Ipa System Design. It consists of an offline phase for model profiling and an online phase for adaptive inference serving.

- **Constructing the query** All queries are constructed based on a template given as:- $InputData \rightarrow Filters \rightarrow MLModels \rightarrow SpecializedTask \rightarrow Result.Queryconstructioninvlovesselect$
- **Configuring the query**:- After operators are selected, now we have to select *configuration knobs* in each operator. The configuration must satisfy the accuracy and latency requirement given by the user. This is done using Bayesian Optimization.
- **Online Adaption**:- Vulcan monitors utility (i.e, latency, accuracy, outputsize, bandwidth) change, to detect runtime dynamics (i.e, content, network). To obtain real-time ground truth on query accuracy given unlabelled live data, a dedicated query with the most expensive configuration is launched in the cloud.

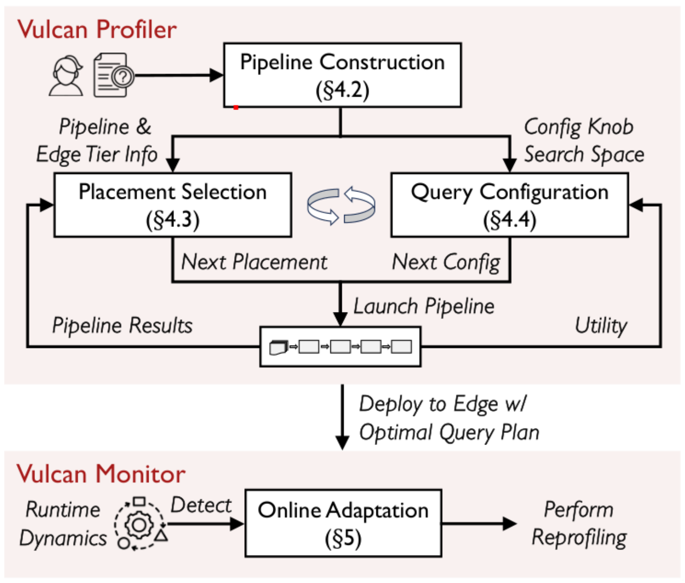


Figure 3.19: Vulcan Architecture

3.1.20 Nvidia Deepstream [27]

Deepstream is commercial product built by Nvidia to help engineers build computer vision queries. It includes library of common operators used in computer vision, such as video decoder/encoder, object detector, object classifier, object tracker. The runtime is built on top of GStreamer[28]. Data model is a patch with metadata, or a batch of patches with metadata. Each operator attaches its result of operation as a metadata in the patch, while using metadata of upstream operators. Deepstream is a single node system allowing it to pass patches by reference (No copy), making it extremely fast, allowing real-time analytics.

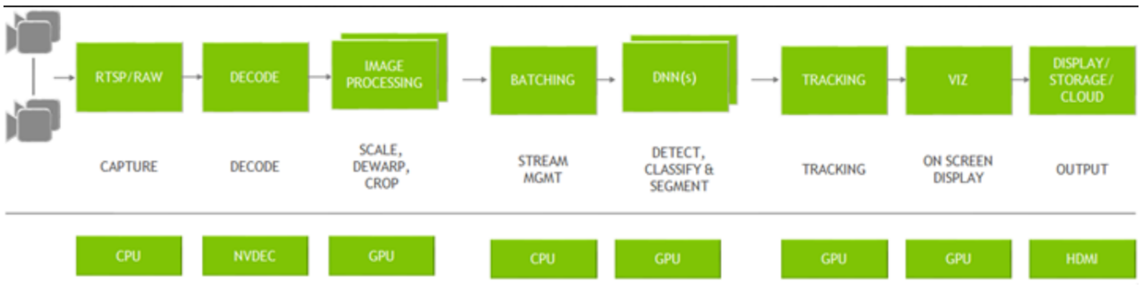


Figure 3.20: Nvidia Deepstream example query

Category	**Systems**
End-to-End Systems	Optasia, Jellybean, DeepLens, Viva, VQPy, IPA
Operator Specific Optimization	NoScope (Proxy Model (PM), Detection), Focus (Index, Classification), Tahoma (PM, Classification), BlazeIt (PM, aggregate, limit), Seiden (Index, aggregate), Miris (Index, Tracking), Otif (Index, Tracking)
Distributed Deployment	Chameleon, VideoEdge, DiStream, JellyBean, IPA
Unbounded Vocabulary	Panorama, Zelda

Table 3.1: Related Work Categorization

4 Research Problems

In this section, we formulate research problems we will try to solve as part of the thesis work.

4.1 Research Problem 01: Predicting end-to-end accuracy of a visual query

4.1.1 Motivation

Accuracy estimation of a physical plan is crucial to be able to pick an optimal plan based on overall accuracy of the query. Typically, accuracy statistics of an individual model is known beforehand, or can be calculated quickly over a validation dataset provided by the user. However, estimating end-to-end accuracy still remains a challenge, which is usually solved by profiling. Estimating end-to-end accuracy is useful when we are enumerating physical plans for a given logical plan, and we want to reject certain plans if they do not meet the accuracy Service Level Agreement (SLA). The objective of research work will be to estimate end-to-end accuracy of query given accuracy of individual models. We want the estimated accuracy to be as close to the actual accuracy of the query.

4.1.2 Related Work

- **Profiling all possible plans**:- Trivially, all possible plans can be enumerated and profiled on a subset of data to get approximate the accuracy. This however does not scale due to exponentially many possible plans and profiling of a single expensive query can take minutes.

- **Profiling with independence assumption**:- If we assume that every configuration knob across all the operators in the query are independent of each other then we can avoid exponential search, by profiling each knob independently while setting other knobs at their 'golden' value. This makes the search linear in number of knobs. This is done in Chameleon[17].

- **Approximating relative orderings of the plans**:- In IPA[11] users specify an upper limit on latency and ask for most accurate query. In this case we do not need to compute exact accuracy if relative ordering is preserved. Relative ordering can be inferred by just multiplying accuracies of individual models. In Jellybean[39], each operator is profiled while varying its parent models. For e.g., consider a model with two inputs which exhibits the following accuracy profile: (60%,50%) $\rightarrow$ 55%, (50%,60%) $\rightarrow$ 60%, and (70%,90%) $\rightarrow$ 65%, then we can conclude that output accuracy is at least 60% if input accuracy is (55%,83%). These methods work fine if SLA is on latency, but if we are to support accuracy based SLA we have to approximate accuracy as close to actual accuracy as possible.

4.1.3 Preliminary Efforts

We are attempting to use Bayesian Network (BN) to model the uncertainty of individual operators and then using the network to estimate the accuracy of the entire query. This requires solving two challenges:-

- Given the logical plan how to generate the BN?
- Given the BN what is the correct expression of accuracy the user want to optimize?

We demonstrate two preliminary experiments. In experiment-1, the BN is easy to synthesize, however the expression for the accuracy is not as straightforward. In experiment-2, the synthesis of BN is non-trivial.

Experiment 1

We create a simple object detection followed by object classification query.

$$\mathit{image} \rightarrow \mathit{DigitDetector} \rightarrow \mathit{DigitClassifier} \rightarrow \mathit{digits}.$$

The image contains one or more digits, *DigitDetector* extract the digit bounding box from the image, and *DigitClassifier* figures out the digit in the bounding box. Accuracy metric for detector is $IoU > 0.7$, which means if $IoU(\mathit{predicted\_bbox}, \mathit{gt\_bbox}) > 0.7$, then it will be considered a correct detection. For classifier the accuracy is simply $\mathit{predicted\_class} == \mathit{gt\_class}$. Table4.1 and 4.2 shows the accuracy of detector and classifier respectively. Table4.3 shows the accuracy of end-to-end query. We first try to estimate end-to-end accuracy by simply multiplying individual accuracy as done in [11], i.e:-

$$\mathit{pipe\_acc} = P(IoU > 0.7) * P(CiC)$$

Here CiC means 'Classifier is Correct'.

However, we see it is not a close approximation, and it is well below the actual accuracy. Explanation for this is that our correctness criteria for detector is too strict, there are few cases where IoU is well below 0.7 and classifier is still able to classify the digit correctly. But reducing the threshold to say, 0.5 doesn't help either, accuracy does move closer to actual accuracy but not by much, and relative ordering is not preserved anymore.

Instead of simply multiplying we could create a BN shown in Fig4.1. The construction implies non-independence between classifier and detector. The intuition is that classifier's accuracy will be dependent on the IoU of detector's output, i.e. higher the IoU more accurate is the classifier. The accuracy of the pipeline can be expressed as:-

$$\begin{aligned} \mathit{pipe\_acc} &= P(IoU < 0.3) * P(CiC|IoU < 0.3) + \\ &= P(0.3 < IoU < 0.5) * P(CiC|0.3 < IoU < 0.5) + \\ &= P(0.5 < IoU < 0.7) * P(CiC|0.5 < IoU < 0.7) + \\ &= P(0.7 < IoU < 0.9) * P(CiC|0.7 < IoU < 0.9) + \\ &= P(0.9 < IoU) * P(CiC|0.9 < IoU) \end{aligned} \tag{4.1}$$

As we can see this requires more parameters (8) whereas with the independence assumption we only needed (2). The parameters for detectors can be calculated trivially. For classifier, we could fuzz the ground truth at different IoUs to get accuracy at different IoUs.

Experiment 2

In this experiment we show that two different physical query plans with different semantics can be used for same logical query. The query is inspired from Deeplens and is as follows:- Find all matching pedestrians from a set of detected objects, *i.e., Pedestrian ReID.*

The two physical plans are as follows:-

Detector Name	**MAP-50**	**MAP-70**
yolov8-n	0.88	0.53
yolov8-s	0.90	0.551
yolov8-m	0.912	0.559
yolov8-l	0.911	0.561

Table 4.1: Trained Detector Accuracy

Classifier Name	**Accuracy-Top1**
yolov8-cls-n	0.87
yolov8-cls-s	0.88
yolov8-cls-m	0.0.89
yolov8-cls-l	0.0.9

Table 4.2: Trained Classifier Accuracy

Detectors	Classifier	Actual Accuracy	Estimated Accuracy @MAP50	Estimated Accuracy @MAP70
yolov8-n	yolov8-cls-n	0.68	0.59	0.46
yolov8-n	yolov8-cls-s	0.69	0.61	0.47
yolov8-n	yolov8-cls-m	0.7	0.63	0.48
yolov8-n	yolov8-cls-l	0.7	0.64	0.485
yolov8-s	yolov8-cls-n	0.71	0.62	0.48
yolov8-s	yolov8-cls-s	0.72	0.64	0.488
yolov8-s	yolov8-cls-m	0.73	0.65	0.494
yolov8-s	yolov8-cls-l	0.74	0.67	0.499
yolov8-m	yolov8-cls-n	0.749	0.63	0.488
yolov8-m	yolov8-cls-s	0.76	0.65	0.496
yolov8-m	yolov8-cls-m	0.77	0.67	0.502
yolov8-m	yolov8-cls-l	0.77	0.68	0.507
yolov8-l	yolov8-cls-n	0.75	0.63	0.489
yolov8-l	yolov8-cls-s	0.76	0.65	0.497
yolov8-l	yolov8-cls-m	0.778	0.66	0.503
yolov8-l	yolov8-cls-l	0.785	0.68	0.508

Table 4.3: End-to-End query Accuracy

- **Filter-Match** (i) Detect all objects (ii) Filter all pedestrians (iii) Match them with each other using cosine similarity of feature vectors. *i.e.,*

  $$isMatching(O_1, O_2) \land isHuman(O_1) \land isHuman(O_2)$$

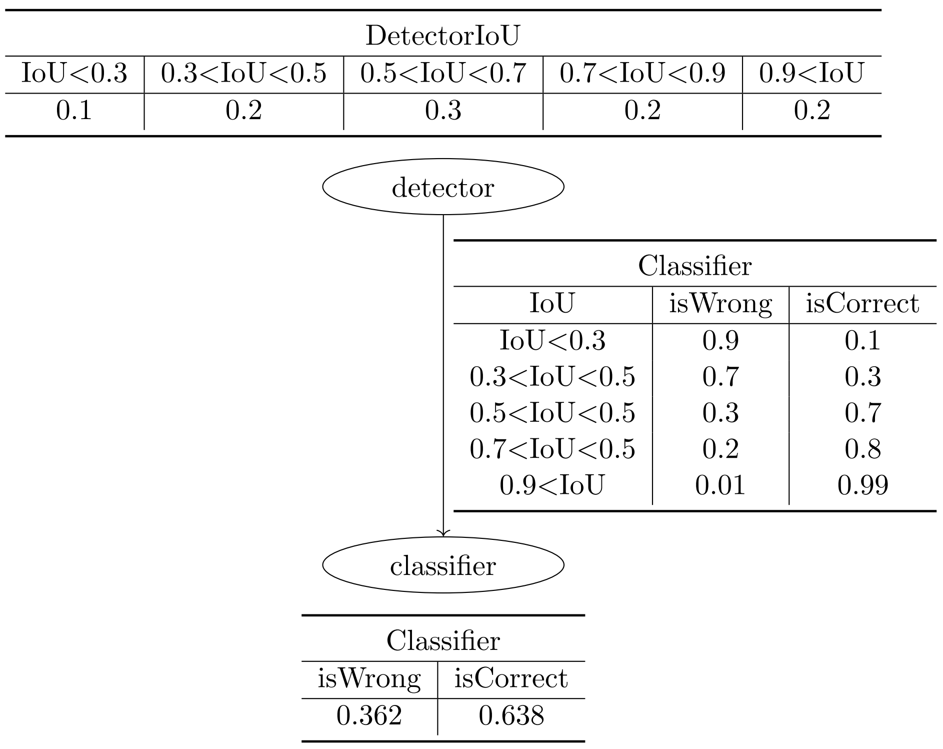
- **Match-Filter** (i) Detect all objects (ii) Match all objects (iii) If either of the matching objects are pedestrian then consider them similar pedestrian. *i.e.,*

  $$isMatching(O_1, O_2) \land (isHuman(O_1) \lor isHuman(O_2))$$

  Note that the assumption here is that a less precise filter can misclassify pedestrian as a non-pedestrian, but the pedestrian matcher will not match pedestrian with some other object.

We model the above two plans using BN (fig4.2) and check if it matches the result mentioned in the paper (it did). Both plans have the same BN except the 'isSame' node. Brief description of the nodes in the BN is follows:-

- gtIsHuman1 & gtIsHuman2:- represent the probability that two objects being compared are actually humans. This is the property of the dataset. They have identical distribution, but they represent two different variables.
- gtIsMatching:- represents the probability that two objects are actually matching. We know that a human and a non-human can never be matching hence the distribution also reflects that. This is also the property of the dataset.



DetectorIoU				
IoU<0.3	0.3<IoU<0.5	0.5<IoU<0.7	0.7<IoU<0.9	0.9<IoU
0.1	0.2	0.3	0.2	0.2

Classifier		
IoU	isWrong	isCorrect
IoU<0.3	0.9	0.1
0.3<IoU<0.5	0.7	0.3
0.5<IoU<0.5	0.3	0.7
0.7<IoU<0.5	0.2	0.8
0.9<IoU	0.01	0.99

Classifier	
isWrong	isCorrect
0.362	0.638

Figure 4.1: Bayesian Network For Experiment-1. The table in the bottom is the overall (theoretical) accuracy of the query.

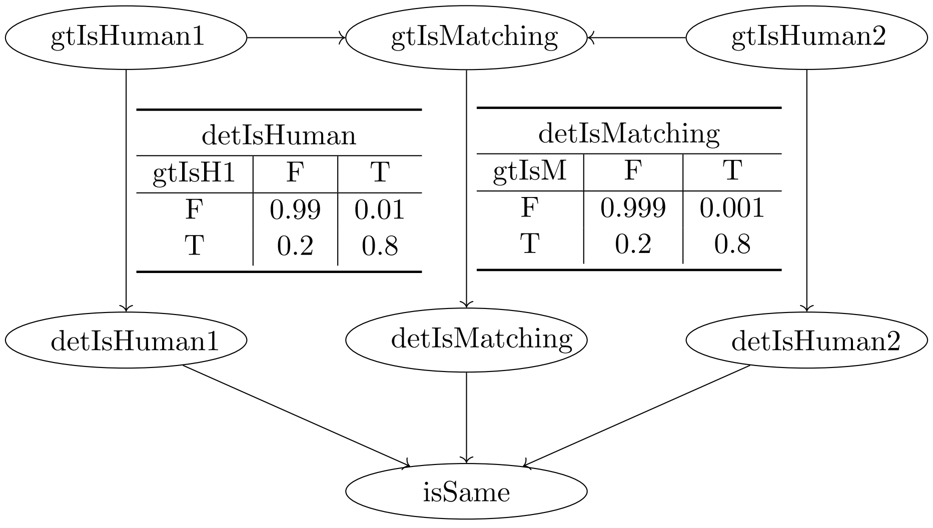
- detIsHuman1 & detIsHuman2:- represent the probability that the ML model will detect the human/non-human given the ground truth is human/non-human. This is the property of the ML model.
- detIsMatching:- represents the probability that the model will say that two objects are matching/non-matching given the ground truth is matching/non-matching. This is also the property of the ML model.
- isSame:- represents the core query logic of the query. This is the variable where two plans disagree. The FM plan(left table) outputs True only if all 3 inputs are True, whereas the MF plan(right table) outputs True in 3 cases. This results in higher recall for the MF plan.

The plan 1 is faster (it has to process less frames due to the filter). However, plan 2 has higher recall if the filter is less precise (filter will make mistakes, but matcher will correct them). This can be useful if we do not want to miss any positive matches, but are fine with false matches.

In fig 4.3 we plot the precision/recall of the two queries against accuracy of ML model. In the left plot we vary the precision of the human detection model and observe the precision/recall of the end-to-end query. In the right plot we vary the precision of the matching model. We observe that precision of FM plan is always higher than MF plan, whereas recall of MF plan is always higher than FM plan. Using this plot we can derive precision requirement of the ML models given precision/recall/latency requirement of the query.

gtIsHuman	
F	T
0.7	0.3

gtIsMatching			
gtIsH1	gtIsH2	F	T
F	F	0.9	0.1
F	T	1	0
T	F	1	0
T	T	0.9	0.1



detIsHuman		
gtIsH1	F	T
F	0.99	0.01
T	0.2	0.8

detIsMatching		
gtIsM	F	T
F	0.999	0.001
T	0.2	0.8

isSame				
dIsH1	dIsH2	dIsM	F	T
F	F	F	1	0
F	F	T	1	0
F	T	F	1	0
F	T	T	1	0
T	F	F	1	0
T	R	T	1	0
T	T	F	1	0
T	T	T	0	1

isSame				
dIsH1	dIsH2	dIsM	F	T
F	F	F	1	0
F	F	T	1	0
F	T	F	1	0
F	T	T	0	1
T	F	F	1	0
T	F	T	0	1
T	T	F	1	0
T	T	T	0	1

Figure 4.2: BN for experiment-2. Here $gtIsHuman1$ (Probability that the object being matched is human), $gtIsHuman2$ and $gtIsMatching$ (Probability that any two objects are same) are properties of dataset. $gtIsHuman1$ and $gtIsHuman1$ have identical distribution. $detIsHuman1$ (Probability that the human is detected), $detIsHuman2$ and $detIsMatching$ (Probability that objects are matched) are properties of the ML models. FM_isSame and MF_isSame are the deterministic variable and the final output of query filter-match, and match-filter query respectively.

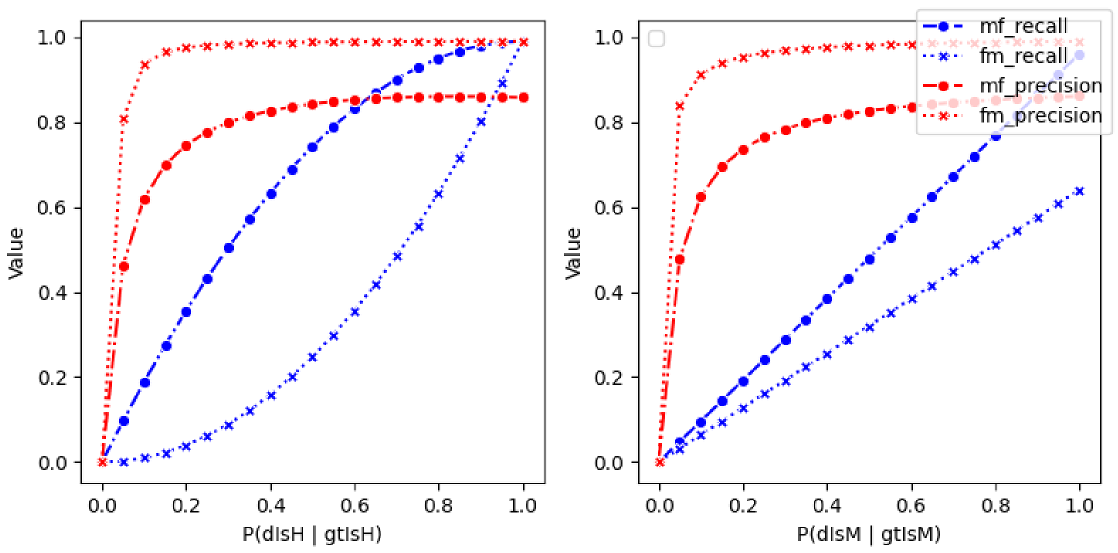


Figure 4.3: Precision/Recall of query vs Accuracy of ML models

4.2 Research Problem 02: Automatic visual query synthesis

4.2.1 Motivation

As logical operators become higher-level, the space of possible physical operators grows exponentially, making them tedious to specify. Specifying all the rules is necessary to find optimal physical plans. For example, suppose we have a library of vehicle detection pretrained models. Now suppose use wants to extract trucks, and there exists a truck classifier, a smart synthesizer could stitch vehicle detector and truck classifier together to work as truck detector. This of course requires a domain knowledge specified by the user that truck is a subclass of vehicle. A type system can be built to capture such semantics. We want to create a framework for automatically synthesizing complex visual queries, from minimal specification from user.

4.2.2 Related Work

Recently, type-based program synthesis [29, 13, 10] has shown good promise on synthesizing programs from their specifications. We will explore these techniques by treating logical operators as specifications to automatically synthesize implementation rules that map logical operators to physical operators.

4.2.3 Preliminary Efforts

Exploring Apache Calcite [5]

Apache calcite is a (Java) framework that is mainly used for two things:-

- As a **frontend layer** for custom data storage solution. Suppose, we have built our custom data storage solution and want to create a SQL interface for querying it. This will involve creating a lexer, an AST parser, conversion to relational algebra, and finally the execution layer along with optimizations. Using calcite we can avoid building the entire thing from scratch, get the SQL interface by only implementing certain interfaces provided by the framework.
- As a framework for writing custom **Query Optimization (QO)** rules. Using calcite we have to specify our custom QO rules (along with cost heuristics) and calcite will take care of optimization using its inbuilt optimization engine.

Figure4.4 outlines the main components of Calcite's architecture. Calcite optimizer uses a tree of relational operators as its internal representation. The optimization engine primarily consists of three consists of components:- (i) **Rules** are applied on the relational expression repeatedly to optimize the expression. Some rules are provided by Calcite, and user defined rules can also be added. (ii) **Metadata providers** provides the information regarding the cost of the plan. For example, number of rows in a relation. (iii) **Planner engine**:- contains the core logic of optimization. Two optimizers are provided by Calcite and user defined planners can also be added and used.

Listing 4.1 shows a Filter push down rule expressed in Calcite. In this rule the filter predicate is pushed down in the join to save cost. The interface 'FilterIntoJoinRuleConfig' defines the pattern which when matched will trigger the logic defined in 'onMatch'. Some code has been omitted for conciseness. The pattern catches all the 'filter' that has a single child of type 'join'. 'perform()' is where the filter is actually pushed down the join.

```java
public static class JoinConditionPushRule extends
      FilterJoinRule{
     @Override public void onMatch(RelOptRuleCall
       call) {
       Filter filter = call.rel(0);
       Join join = call.rel(1);
       perform(call, filter, join);
     }
     /** Rule configuration. */
     @Value.Immutable(singleton = false)
     public interface FilterIntoJoinRuleConfig
       extends FilterJoinRule.Config {
        FilterIntoJoinRuleConfig DEFAULT =
             ImmutableFilterIntoJoinRuleConfig.of((
       join, joinType, exp) -> true)
                  .withOperandSupplier(b0 ->
                      b0.operand(Filter.class).
       oneInput(b1 ->
                           b1.operand(Join.class).
       anyInputs()))
                  .withSmart(true);
        @Override default FilterIntoJoinRule toRule
       () {
           return new FilterIntoJoinRule(this);
        }
     }
}
```

Listing 4.1: Filter push down rule expressed in Calcite

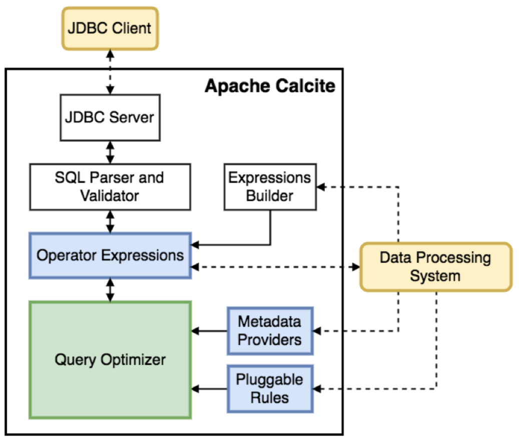


Figure 4.4: Calcite Framework

We want to use Apache Calcite for writing our optimization rules, but we note that it is built for SQL operations on relation data model. However, the framework is very modular, and we can plug in our custom data model specific to visual queries.

Exploring refinement types using Synquid[29]

A **refinement type** is a type augmented with additional predicate which must be held for all elements of the refined type. The types can be defined as $v : B|P(v)$, where B is the base type, and P is a predicate on values of B. For example all natural number less than 5 can be written as $\{n \in \mathbb{N}|n < 5\}$. Refinements can be leveraged by a program synthesizer to efficiently generate a program that satisfies the requirements stated by the type signature. For example in listing 4.3 Syquid generates the implementation of the function that given a natural number n and a value x of type a returns a list of length n containing x.

```haskell
-- Type synonym for natural numbers (_v denotes the "value variable")
upe Nat = {Int | _v >= 0}

-- List data type: just like in Haskell
data List a where
  Nil :: List a
  Cons :: x: a -> xs: List a -> List a

dec :: x: Int -> {Int | _v == x - 1}

-- Our synthesis goal: a function that returns 'n' copies of 'x'
replicate :: n: Nat -> x: a -> {List a | len _v == n}
replicate = ??
```

```

-- Synthesized program:-
{-
replicate = \n . \x .
  if n <= 0
    then Nil
  else
    Cons x (replicate (dec n) x)
-}
```

Listing 4.2: Synquid program to synthesize a funtion, given a specification using type refinement

4.2.4 Tentative Approach

Writing logical and physical transformation rules in Calcite is exhausting. People have manually written 100s of transformation rules within Calcite and in projects using Calcite (like Flink). Each transformation rule takes 100s of lines of code to write. Something concise and declarative would be preferred. It will be interesting to automatically synthesize all these manually written rules.

The main idea is to capture all the Calcite behaviors into a unified type system. This type system facilitates writing rules directly from the subtyping relations within the type system. Relational processing can be expressed as a DAG of relational operators. Relations are flowing on the edges between the operators. Each relation on this edge adheres to a Schema. In general, the operators can:-

- Add new columns, remove columns (projection).
- Enforce new properties on a column (filter).
- Enforce new properties on pair of rows (distinct).

We posit that all of these can be captured directly into a type system. We propose that typed relations flow on the edges of a dataflow Directed Acyclic Graph (DAG). Typed relations subsume schema of the relation and have many more useful properties. We take inspiration from Synquid and define a *refined relation* type.

```
data RefinedRelation <A:: Row -> bool> <B:: Row -> Row -> bool> where
  Nil:: RefinedRelation <A> <B>
  Cons:: {r | Ar} -> {RefinedRelation <A> <B> | B r v} ->
            RefinedRelation <A> <B>

```

Listing 4.3: Refined Relation

Basically, a $RefinedRelation$ is just a list (i.e, rows have a total order among them) with its type specified using two predicates:

- A that takes a row and specifies whether this row can be a part of this relation; and
- B that takes any two rows 'r1' and 'r2' such that row 'r1' is before 'r2' in the relation and specifies whether these two rows can be a part of this relation.

A row is essentially just a dictionary that maps column names to values. Columns within a row can be accessed using . operator. We now write **RefinedRelation** as RR for brevity. Some examples:

- A relation with columns *img_w* and *img_h* each of type integer.

  $$RR\ [A(r) :: r.img\_w \in Z \wedge r.img\_h \in Z]\ [B(r_1, r_2) :: True]$$

  .

- Column *img_id* is primary key.

  $$RR\ [A(r) :: r.img\_id \in Z]\ [B(r_1, r_2) :: r_1.img\_id \neq r_2.img\_id]$$

- We can further refine the types of the columns. Let us say RR goes through $\sigma_{class='truck'}$, output RR will have:

$$RR\ [A(r) :: r.class \in Z \wedge r.class =' truck']\ [B(x, y) :: True$$

- Calcite's calling conventions can be expressed with the predicate A, using 'hidden column' names like _device

$$RR\ [A(r) :: r.\_device = GPU]\ [B : \ldots]$$

- Relationships between columns can also be expressed with the predicate A. For example, column a is derived from column b.

$$RR\ [A(r) :: r.a \in Z \wedge r.b \in Z \wedge r.a = 2 * r.b]\ [B : \ldots]$$

We will explore building refined relation based optimizer and use VAQs as a case study.

4.3 Research Problem 03: Runtime adaption of a visual query in a distributed setting

4.3.1 Motivation

For streaming tasks, the optimal query may not be optimal after a while. For example change in weather condition may hurt accuracy of the query, or execution of some other query on the same hardware may reduce latency. In these cases the query must adapt to satisfy its SLA.

Adaptation can be performed in two different ways:-

- **Changing operator configuration**:- For example, loading a different model, or change in image resolution, batch size.
- **Changing operator placement**:- For example, moving an operator from edge to cloud or vice-versa depending on hardware load.

We want to build a system that adapts a running query according to change in data distribution and hardware load/availability.

4.3.2 Preliminary effort

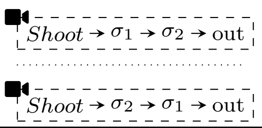
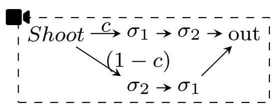
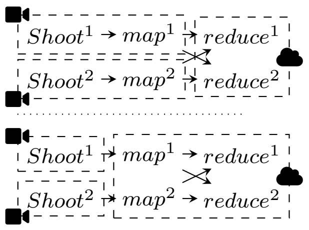
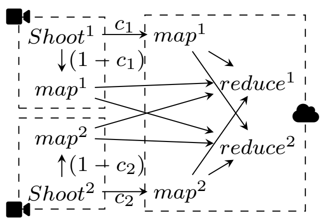
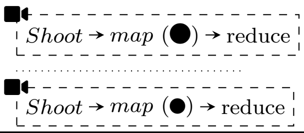
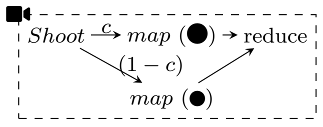
Exploring Nvidia's DeepStream

We explored Nvidia's Deepstream as an physical plan execution engine for visual analytics query. We made the following observations:-

- It is extremely fast. It easily delivers 500fps on a detection tasks.
- It is easy to write a query with existing operators using a declarative syntax.
- However, new operators have to be written in C using Gstreamer and CUDA library, which has learning curve and slows down the development cycle.
- Existing operators are closed source hence we cannot perform any experiments that require us to change the operator itself, and writing one from scratch is hard.
- Although we can send query metrics to cloud, it is not a distributed system all video processing happens on a single node.

We concluded that we Deepstream is not research friendly, however we should keep an eye on its progress as a commercial software.

Table 4.4: Examples of flexible physical plans

Example	**Physical plans**	**Flexible physical plan**
$vdo\_in \to \sigma_1 \to \sigma_2 \to$ out		
$vdo\_in \to map \to reduce$	 Shoot¹ → map¹ reduce¹ Shoot² → map² reduce² Shoot¹ map¹ → reduce¹ Shoot² map² → reduce²	 Shoot¹ c1 map¹ (1 − c1) map¹ reduce¹ map² reduce² (1 − c2) Shoot² c2 map²
$vdo\_in \to map$ (●/●) $\to$ reduce		

4.3.3 Tentative Approach

Since each VAP can have multiple physical plans, the problem of adaptation is how to switch from one physical plan to another to keep satisfying the accuracy and latency SLOs while minimizing the cloud bill. It is important to keep the switches between plans gradual with the possibility of quickly rolling back switches if the new plan does not work as expected. We build a *Flexible Physical Plan (FPP)* that uncovers a broader space of trade-offs and directly facilitates adaptation. FPPs combine multiple complementary physical plans. Compared to using any fixed physical plan, using a flexible physical plan allows gradually moving from one physical plan to another and opens up the space of available physical plans by including mixtures of complementary physical plans. Our key insight is that these mixed plans can provide better trade-offs than individual physical plans, as illustrated by the alternative models and operator placement VAP examples below.

(1) Commutative filters. The first example in Table shows a simple query with two filters denoted by σ_1 and σ_2. For example, we might want to find out when the traffic signal is red (σ_1), and then find out if there are cars on the intersection (σ_2). Since filters can be run in any order, there are two physical plans for this dataflow graph, each running a different permutation of filters.

The flexible physical plan merges these two physical plans into a combined graph. In the combined graph, FPP adds a probability on each edge when it combines the graph: with probability c, the operator *Shoot* sends data (e.g, video frames) to σ_1 (e.g, is the light red?) and with probability $(1 - c)$, it sends data to σ_2 (e.g, is there a vehicle on intersection?). It is clear that switching c from 0 to 1 and vice-versa changes from one physical plan to another.

Adaptation might indeed change c depending on the incoming data. For example, if the light has just turned green, it is better to set $c = 1$, i.e, check for the light being red to quickly discard the frame. Once the light turns red, it is better to set $c = 0$, i.e, first find cars on the road and then verify that the light is red. FPPs allow easily reasoning about different adaptation logic since the adaptation controller just needs to control c. For example, if we already know the light switching schedule at the intersection, then we can just change the value of c based on the schedule. If we have no knowledge of the schedule, we might slowly increase c, when the light is red, to sample some frames in case the light has changed to green.

(2) Operator placement. The first example in Table shows a simple map reduce query, *e.g.,*run object detection on each video frame and count the number of vehicles of each type in 5-minute time windows. There are multiple physical plans depending on the operator placement, such as placing `map` on the cameras or placing it on the cloud. Existing literature on adapting VAPs typically choose one of the two plans. However, neither of the two plans may be optimal. If we place the `map` function on the cameras, the cameras might run out of compute resources thereby not being able to keep up with the frame rate. If we place the `map` function on the server, the camera CPUs might be underutilized and the camera network might start becoming the bottleneck.

These physical plans are merged into a combined FPP while adding probabilities of running appropriate `map` functions. This increases the exploration space of doing adaptation. For example, by setting c_2 to 0.3, camera-2 can offload `map` function for 30% of its frames and process the rest locally.

(3) Alternative models. The next example in Table again shows a simple map reduce query. Here, the `map` function has multiple available implementations denoted by ● and ●, *e.g.,*heavyweight and lightweight object detection models. Hence, there are multiple physical plans: one for each available implementation of `map`.

Again, existing video analytics research switch completely to cheaper implementation, *e.g.,*using the lightweight model, or to the costly implementation, *e.g.,*using the heavyweight model. However, neither of these may uphold the required SLOs. The cheaper plan may satisfy the latency SLO but not the accuracy SLO whereas the costly plan may satisfy the accuracy SLO but not the latency SLO. By setting $c = 0.4$, we can easily run a light weight model for 60% of the frames and run the rest 40% of the frames on the heavy weight model. Doing so opens new design points to satisfy both accuracy and latency SLOs.

5 Conclusion

Video analytics has emerged as a transformative technology with applications spanning various domains such as surveillance, healthcare, retail, and transportation. However, current platforms do not provide satisfactory developer experience and development of new video analytics query is still mostly adhoc. A developer writing a SQL query does not need to worry about the underlying execution plans or optimizations (most of the time). A developer writng a visual query should have a similar experience. In this work, we examined the landscape of video analytics systems, and outlined the work that is to be done as part of the future research work. Our study has identified three key research problems: (i) Estimating end-to-end accuracy of a VAQ, (ii) synthesis of a VAQ, and (iii) automatic runtime adaption of a VAQ.

References

[1] Michael R. Anderson et al. “Physical Representation-Based Predicate Optimization for a Visual Analytics Database”. In: *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 2019 IEEE 35th International Conference on Data Engineering (ICDE). Macao, Macao: IEEE, Apr. 2019, pp. 1466–1477. ISBN: 978-1-5386-7474-1. DOI: 10.1109/ICDE.2019.00132. URL: https://ieeexplore.ieee.org/document/8731447/ (visited on 06/21/2024).

[2] Jaeho Bang et al. “Seiden: Revisiting Query Processing in Video Database Systems ”. In: *Proceedings of the VLDB Endowment* 16.9 (May 2023), pp. 2289–2301. ISSN: 2150-8097. DOI: 10.14778/3598581.3598599. URL: https://dl.acm.org/doi/10.14778/3598581.3598599 (visited on 06/21/2024).

[3] Favyen Bastani and Samuel Madden. “OTIF: Efficient Tracker Pre-processing over Large Video Datasets”. In: *Proceedings of the 2022 International Conference on Management of Data*. SIGMOD/PODS ’22: International Conference on Management of Data. Philadelphia PA USA: ACM, June 10, 2022, pp. 2091–2104. ISBN: 978-1-4503-9249-5. DOI: 10.1145/3514221.3517835. URL: https://dl.acm.org/doi/10.1145/3514221.3517835 (visited on 06/21/2024).

[4] Favyen Bastani et al. “MIRIS: Fast Object Track Queries in Video”. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. SIGMOD/PODS ’20: International Conference on Management of Data. Portland OR USA: ACM, June 11, 2020, pp. 1907–1921. ISBN: 978-1-4503-6735-6. DOI: 10.1145/3318464.3389692. URL: https://dl.acm.org/doi/10.1145/3318464.3389692 (visited on 06/21/2024).

[5] Edmon Begoli et al. “Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources”. In: *Proceedings of the 2018 International Conference on Management of Data*. 2018, pp. 221–230.

[6] John Canny. “A Computational Approach To Edge Detection”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* PAMI-8 (Dec. 1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[7] Z. Cao et al. “OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019).

[8] Ronnie Chaiken et al. “Scope: easy and efficient parallel processing of massive data sets”. In: *Proceedings of the VLDB Endowment* 1.2 (2008), pp. 1265–1276.

[9] Liang-Chieh Chen et al. “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs”. In: *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2017), pp. 834–848.

[10] Jonas Fiala et al. “Leveraging Rust Types for Program Synthesis”. In: 7.PLDI (2023), pp. 1414–1437.

[11] Saeid Ghafouri et al. “IPA: Inference Pipeline Adaptation to Achieve High Accuracy and Cost-Efficiency”. In: *arXiv preprint arXiv:2308.12871* (2023).

[12] Ross Girshick. “Fast r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[13] Zheng Guo et al. “Type-directed program synthesis for RESTful APIs”. In: (2022), pp. 122–136.

[14] Kaiming He et al. “Mask R-CNN”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.2 (2020), pp. 386–397. DOI: 10.1109/TPAMI.2018.2844175.

[15] Paul VC Hough. *Method and means for recognizing complex patterns*. US Patent 3,069,654. Dec. 1962.

[16] Kevin Hsieh et al. “Focus: Querying Large Video Datasets with Low Latency and Low Cost”. In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 2018, pp. 269–286. URL: https://www.usenix.org/conference/osdi18/presentation/hsieh (visited on 06/21/2024).

[17] Junchen Jiang et al. “Chameleon: Scalable Adaptation of Video Analytics”. In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM ’18: ACM SIGCOMM 2018 Conference. Budapest Hungary: ACM, Aug. 7, 2018, pp. 253–266. ISBN: 978-1-4503-5567-4. DOI: 10.1145/3230543.3230574. URL: https://dl.acm.org/doi/10.1145/3230543.3230574 (visited on 06/21/2024).

[18] Daniel Kang, Peter Bailis, and Matei Zaharia. *BlazeIt: Optimizing Declarative Aggregation and Limit Queries for Neural Network-Based Video Analytics*. Dec. 9, 2019. arXiv: 1805.01046 [cs]. URL: http://arxiv.org/abs/1805.01046 (visited on 06/21/2024). preprint.

[19] Daniel Kang et al. *NoScope: Optimizing Neural Network Queries over Video at Scale*. Aug. 8, 2017. arXiv: 1703.02529 [cs]. URL: http://arxiv.org/abs/1703.02529 (visited on 06/21/2024). preprint.

[20] Anthony Kay. “Tesseract: an open-source optical character recognition engine”. In: *Linux J.* 2007.159 (July 2007), p. 2. ISSN: 1075-3583.

[21] Alex Kendall, Matthew Grimes, and Roberto Cipolla. “PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015.

[22] Sanjay Krishnan, Adam Dziedzic, and Aaron J. Elmore. *DeepLens: Towards a Visual Data Management System*. Dec. 18, 2018. arXiv: 1812.07607 [cs]. URL: http://arxiv.org/abs/1812.07607 (visited on 06/07/2024). preprint.

[23] Junnan Li et al. “Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation”. In: *International conference on machine learning*. PMLR. 2022, pp. 12888–12900.

[24] Minghui Liao et al. *Real-Time Scene Text Detection with Differentiable Binarization and Adaptive Scale Fusion*. 2022. arXiv: 2202.10304 [cs.CV].

[25] Wei Liu et al. “Ssd: Single shot multibox detector”. In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam , The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer. 2016, pp. 21–37.

[26] Yao Lu, Aakanksha Chowdhery, and Srikanth Kandula. “Optasia: A Relational Platform for Efficient Large-Scale Video Analytics”. In: *Proceedings of the Seventh ACM Symposium on Cloud Computing*. SoCC ’16: ACM Symposium on Cloud Computing. Santa Clara CA USA: ACM, Oct. 5, 2016, pp. 57–70. ISBN: 978-1-4503-4525-5. DOI: 10.1145/2987550.2987564. URL: https://dl.acm.org/doi/10.1145/2987550.2987564 (visited on 01/06/2024).

[27] Nvidia. *Nvidia Deepstream*. 2024. URL: https://docs.nvidia.com/metropolis/deepstream/dev-guide/index.html (visited on 06/21/2024).

[28] OpenSource. *GStreamer*. 2024. URL: https://gstreamer.freedesktop.org/documentation (visited on 06/21/2024).

[29] Nadia Polikarpova, Ivan Kuraj, and Armando Solar-Lezama. "Program synthesis from polymorphic refinement types". In: *ACM SIGPLAN Notices* 51.6 (2016), pp. 522–538.

[30] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV]. URL: https://arxiv.org/abs/2103.00020.

[31] Joseph Redmon and Ali Farhadi. "Yolov3: An incremental improvement". In: *arXiv preprint arXiv:1804.02767* (2018).

[32] Francisco Romero et al. "Optimizing Video Analytics with Declarative Model Relationships". In: *Proceedings of the VLDB Endowment* 16.3 (Nov. 2022), pp. 447–460. ISSN: 2150-8097. DOI: 10.14778/3570690.3570695. URL: https://dl.acm.org/doi/10.14778/3570690.3570695 (visited on 06/21/2024).

[33] Francisco Romero et al. *Zelda: Video Analytics Using Vision-Language Models*. Nov. 7, 2023. arXiv: 2305.03785 [cs]. URL: http://arxiv.org/abs/2305.03785 (visited on 06/21/2024). preprint.

[34] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich , Germany, October 5-9, 2015, proceedings, part III 18*. Springer. 2015, pp. 234–241.

[35] Irwin Sobel et al. "Sobel-feldman operator". In: *Preprint at https://www. researchgate. net/profile/Irwin-Sobel/publication/285159837. Accessed* 20 (2022).

[36] Mingxing Tan and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *International conference on machine learning*. PMLR. 2019, pp. 6105–6114.

[37] Gemini Team et al. *Gemini: A Family of Highly Capable Multimodal Models*. 2024. arXiv: 2312.11805 [cs.CL].

[38] Alexander Toshev and Christian Szegedy. "DeepPose: Human Pose Estimation via Deep Neural Networks". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, June 2014. DOI: 10.1109/cvpr.2014.214. URL: http://dx.doi.org/10.1109/CVPR.2014.214.

[39] Yongji Wu et al. *Serving and Optimizing Machine Learning Workflows on Heterogeneous Infrastructures*. Aug. 3, 2022. arXiv: 2205.04713 [cs]. URL: http://arxiv.org/abs/2205.04713 (visited on 06/25/2024). preprint.

[40] Shan Yu et al. "VQPy: An Object-Oriented Approach to Modern Video Analytics". In: *Proceedings of Machine Learning and Systems* 6 (2024), pp. 279–295. URL: https://proceedings.mlsys.org/paper_files/paper/2024/hash/87eaaa8605a1a472d9a9756e7500517b
Abstract-Conference.html (visited on 06/21/2024).

[41] Sergey Zagoruyko and Nikos Komodakis. "Wide residual networks". In: *arXiv preprint arXiv:1605.07146* (2016).

[42] Haoyu Zhang et al. "Live Video Analytics at Scale with Approximation and {Delay-Tolerance}". In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 2017, pp. 377–392. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zhang (visited on 06/21/2024).

[43] Yiwen Zhang et al. “Vulcan: Automatic Query Planning for Live ML Analytics”. In: *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. Santa Clara, CA: USENIX Association, Apr. 2024, pp. 1385–1402. ISBN: 978-1-939133-39-7. URL: https://www.usenix.org/conference/nsdi24/presentation/zhang-yiwen.

[44] Yuhao Zhang and Arun Kumar. “Panorama: A Data System for Unbounded Vocabulary Querying over Video”. In: *Proceedings of the VLDB Endowment* 13.4 (Dec. 9, 2019), pp. 477–491. ISSN: 2150-8097. DOI: 10.14778/3372716.3372721. URL: https://dl.acm.org/doi/10.14778/3372716.3372721 (visited on 06/21/2024).