

SOFTWARE SYSTEMS FOR LARGE-SCALE RETROSPECTIVE VIDEO ANALYTICS

by

Tiantu Xu

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy



School of Electrical and Computer Engineering

West Lafayette, Indiana

May 2021

**THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL**

Dr. Felix Xiaozhu Lin, Chair

School of Electrical and Computer Engineering

Dr. Saurabh Bagchi

School of Electrical and Computer Engineering

Dr. Y. Charlie Hu

School of Electrical and Computer Engineering

Dr. Xiangyu Zhang

Department of Computer Science

Approved by:

Dr. Dimitrios Peroulis

This thesis is dedicated to my parents and my fiancée.

ACKNOWLEDGMENTS

I want to thank my Ph.D. advisor Prof. Felix Xiaozhu Lin for his patience, encouragement, and guidance in the past five years, which always helped me out whenever I got stuck. I am indebted to Prof. Lin for the precious opportunity he offered to me to conduct computer systems research when I was a student with no related background.

I want to thank Prof. Saurabh Bagchi, Prof. Y. Charlie Hu, and Prof. Xiangyu Zhang for serving on my Ph.D. advisory committee and providing me with insightful comments on my research.

I enjoy working with my colleagues at Crossroads Systems Exploration Lab (XSEL): Hongyu Miao, Liwei Guo, Heejin Park, and Shuang Zhai. I will miss the days hanging out and moments of inspiring discussions with all of you.

I am grateful for the collaborations with Mengwei Xu, Xuanzhe Liu, Yunxin Liu, Gang Huang, Guohui Wang, Humphrey Shi, Yang Fu, Kaiwen Shen, and Luis Fernando Materon Botelho. Their expertise, hard work, and insightful suggestions on my research are invaluable to me.

Finally, I want to thank my parents, Xiangyang Xu and Yuan Lin, and my fiancée Yinong Zhou, for their love and unconditional support.

TABLE OF CONTENTS

LIST OF TABLES	10
LIST OF FIGURES	11
ABSTRACT	14
1 INTRODUCTION	16
1.1 Supporting video storage for cost-efficient retrospective video analytics	17
1.2 Supporting exploratory video queries on low-cost wireless cameras	20
1.3 Supporting object re-identification under city-scale camera deployments	24
2 SUMMARY	29
3 VSTORE: A DATA STORE FOR ANALYTICS ON LARGE VIDEOS	31
3.1 Background & motivations	31
3.1.1 Retrospective video analytics	31
3.1.2 System model	31
3.1.3 Video format knobs	33
3.1.4 Knob impacts	35
3.2 A case for a new video store	37
3.2.1 The configuration problem	37
3.2.2 Inadequacy of existing video stores	39
3.3 The VStore design	40
3.3.1 Overview	40

3.3.2	Configuring consumption formats	42
3.3.3	Configuring storage formats	44
3.3.4	Planning age-based data erosion	46
3.4	Implementation	49
3.5	Evaluation	49
3.5.1	Methodology	50
3.5.2	End-to-end results	51
3.5.3	Adapting to resource budgets	55
3.5.4	Configuration overhead	56
3.6	Discussion	59
3.7	Related work	60
3.8	Conclusions	62
4	DIVA: SUPPORTING EXPLORATORY VIDEO QUERIES ON ZERO-STREAMING CAMERAS	63
4.1	Background & motivations	63
4.1.1	Cold videos are already pervasive	63
4.1.2	Target queries and their execution	63
4.1.3	A case for zero streaming	64
4.2	Overview	65
4.3	Landmark design	68

4.4	Online operator upgrade	70
4.4.1	The rationale	70
4.4.2	Multipass, multi-operator execution	72
4.5	Query execution planning	73
4.5.1	Executing <i>Retrieval</i> queries	73
4.5.2	Executing <i>Tagging</i> queries	75
4.5.3	Executing <i>Counting</i> queries	75
4.6	Implementation and methodology	76
4.7	Evaluation	79
4.7.1	End-to-end performance	79
4.7.2	Validation of query execution design	83
4.8	Related work	86
4.9	Conclusions	87
5	CLIQUE: SPATIOTEMPORAL OBJECT RE-IDENTIFICATION AT THE CITY SCALE	88
5.1	Background & motivations	88
5.1.1	System model	88
5.1.2	Challenge 1: Algorithm limitations	89
5.1.3	Challenge 2: Numerous cameras & videos	90
5.1.4	Why is prior work inadequate	91

5.2	Clique overview	92
5.3	Clustering unreliable bounding boxes	94
5.4	Incremental search in spatiotemporal cells	96
5.4.1	Assessing cell promises	96
5.4.2	Prioritizing cells in search	97
5.4.3	The search process	99
5.5	Optimizations	100
5.5.1	Optimizations with extra knowledge	100
5.5.2	Utilizing cheap vision operators	101
5.6	Evaluation	102
5.6.1	Methodology	103
5.6.2	End-to-end performance	105
5.6.3	Validation of key designs	106
5.6.4	Comparisons to alternative designs	108
5.6.5	Sensitivity to parameters and inputs	110
5.6.6	Delay reduction by processing at ingestion	111
5.6.7	Impact of optimizations	112
5.6.8	Impact of cheaper vision operators	113
5.7	Related work	115
5.8	Conclusions	116

6	CONCLUSIONS AND FUTURE DIRECTIONS	117
6.1	Conclusions	117
6.2	Future directions	117
	REFERENCES	119
	VITA	140

LIST OF TABLES

3.1	Knobs and their values considered in this work. Total: 7 knobs and 15K possible combinations of values. Note: no video quality and coding knobs for RAW. . . .	34
3.2	The library of operators in the current VStore.	40
3.3	A sample configuration of video formats automatically derived by VStore. . . .	51
3.4	In response to ingestion budget drop, VStore tunes coding and coalesces formats to stay under the budget with increase in storage cost. Changed knobs shown in red.	55
4.1	Cheap μ SD cards on cameras retain long videos for humans to review [128] or for machines to analyze [15].	65
4.2	A summary of supported queries. \mathcal{T} is the queried video timespan; \mathcal{C} is the queried object class.	66
4.3	15 videos used for test. Each video: 720P at 1FPS lasting 48 hours. Column 1: video type. T – traffic; O/I – outdoor/indoor surveillance; W – wildlife.	77
4.4	Experiment configurations.	78
4.5	DIVA’s performance (speedup) with various bandwidths. Numbers: min/median/max of times (\times) of query delay reduction compared to baselines (rows). Averaged on all videos and 9 bandwidths in 0.1MB/s–10MB/s.	81
5.1	The augmented video dataset used in evaluation.	104

LIST OF FIGURES

1.1	The design space of existing video software systems [6], [8], [9], [11]–[15]. . .	17
1.2	The VStore architecture, showing the video data path and backward derivation of configuration.	18
1.3	Overview of DIVA.	22
1.4	The classic pipeline for object ReID, formulated as image retrieval.	24
3.1	Video queries as operator cascades [8], [64].	32
3.2	Impacts of coding knobs. Video: 100 seconds from <i>tucson</i> . See Section 3.5 for dataset and test hardware.	35
3.3	Fidelity knobs have high, complex impacts on costs of multiple components (normalized on each axis) and operator accuracy (annotated in legends). Each plot: one knob changing; all others fixed. See Section 3.5 for methodology. .	36
3.4	Disparate costs of fidelity options A–C, despite all leading to operator accuracy ≈ 0.8 . Operator: License. Cost normalized on each axis. See Section 3.5 for methodology.	37
3.5	Video retrieval <i>could</i> bottleneck consumption. This is exemplified by the decoding speed vs. consumption speed comparisons for two different operators. (a) Operator: License. Consumption can be faster than decoding (speed shown as the dashed line), if the on-disk video is stored with the richest fidelity as ingested. Yet, consumption is still slower than decoding video of the same fidelity (white columns). (b) Operator: Motion. Consumption is faster than decoding, even if the on-disk video is of the same fidelity as consumed. Operator accuracy annotated on the top. See Section 3.5 for test hardware.	38
3.6	VStore derives the configuration of video formats. Example consumption/retrieval speed is shown.	41
3.7	Search in a set of 2D spaces for a fidelity option with accuracy ≥ 0.8 and max consumption speed (i.e., min consumption cost).	43
3.8	Iterative coalescing of storage formats.	45
3.9	Data erosion decays operator speed and keeps storage cost under budget. Small cells: video segments.	47
3.10	End-to-end result of VStore.	53
3.11	Transcoding cost does not scale up with the number of operators. Operator sequence follows Table 3.2.	56
3.12	Age-based decay in operator speed (a) and reducing storage cost (b) to respect storage budget.	57

3.13	Time spent on deriving consumption formats. Numbers of profiling runs are annotated above columns. Each required profiling runs on a 10-second video segment. VStore reduces overhead by 5× in total.	58
4.1	The workflow of DIVA’s query execution.	67
4.2	Class spatial skews in videos. In (a) Banff: 80% and 100% of cars appear in regions that are only 19% and 57% of the whole frame, respectively.	68
4.3	Class spatial distribution can be estimated from sparse frames sampled over long video footage. Among the three heatmaps: while sparse sampling over short footage (left) significantly differs from dense sampling of long footage (right), sparse sampling of long footage (middle) is almost equivalent to Video: Tucson (see Table 4.3).	68
4.4	On-camera operators benefit from long-term video knowledge substantially. Each marker: an operator. For querying buses on video Banff (see Table 4.3).	69
4.5	Three alternative executions of a Retrieval query, showing multipass ranking (bottom) outperforms running individual rankers alone (top two). Each row: snapshots of the upload queue at three different moments. In a queue: ranking/uploading frames from left to right.	71
4.6	Cheap/expensive camera operators excel at different query stages. Each sub-figure: two alternative executions of the same query, showing query progress (bars) and the corresponding operator’s progress (arrows).	72
4.7	On <i>Retrieval</i> and <i>Tagging</i> queries, DIVA shows good performance and outperforms the alternatives. x-axis for all: query delay (secs). y-axis for (a): % of retrieved instances; for (b): refinement level (1/N frames).	79
4.8	On <i>Counting</i> queries, DIVA shows good performance and outperforms the alternatives. Legend: see Figure 4.7. x-axis for all: query delay (secs). y-axis for left plots: count; for top two right plots: ground truth for avg/median queries; for bottom right plot: % of max value.	80
4.9	DIVA significantly reduces network traffic compared to “all streaming”. Results averaged over all videos.	82
4.10	DIVA’s both key techniques – optimization with long-term video knowledge (opt) and operator upgrade (upgrade), contribute to performance significantly.	83
4.11	Validation of landmark design. In (a)/(b)/(c): Left – <i>Retrieval</i> on Chaweng; Right – <i>Tagging</i> on JacksonH.	84

5.1	Examples of unreliable bounding boxes. (Left) an image of vehicle A, whose feature is the input. (Top) a histogram of distances between the input and other features of A. (Bottom) a histogram of distances between the input and features of B, a confusing vehicle. All features are 1×1024 vectors extracted by ResNet-152. Euclidean distances with L-2 norm [51] are used. Video clips: 4.7/4.9 sec for vehicle A/B from CityFlow [37].	89
5.2	An overview of Clique. * = a starter camera.	93
5.3	Clustering of bounding boxes tolerates low frame rates. Y-axis: the percentage of bounding boxes correctly attributed to respective objects. Object tracking implemented in OpenCV 3.4.4. Videos from CityFlow [37].	96
5.4	Augmenting real-world city videos [37] as our test dataset: duplicating the original epoch; erasing random vehicles from each epoch; erasing the target vehicle from all but the original epoch.	103
5.5	Query-by-query comparison between Clique and the alternatives, broken down by per-query comparison outcomes. Numbers on bottom: accuracy goals; (X/Y): X = number of queries that Clique reached the accuracy; Y = total query count.	105
5.6	The CDF of query delays by Clique and the alternatives. (X/Y): X = the number of queries on which all the versions reach the accuracy goal; Y = total query count.	107
5.7	The CDF of query delays by Clique, ReXCam-ST, and PROVID-ST. (X/Y): X = the number of queries on which all the versions reach the accuracy goal; Y = total query count.	109
5.8	A comparison between Clique's choices of starter cameras and random choices. (X/Y) in (a): X = number of queries that Clique reached the accuracy; Y = total query count.	110
5.9	Query delays with N cameras per geo-group pre-processed at ingestion time. In case N exceeds the total cameras of a geo-group, all the cameras are pre-processed. (X/Y): X = number of queries on which all versions reach the accuracy goal; Y = the total query count. Y-axis in logscale.	111
5.10	Delay CDFs of Clique augmented to exploit camera orientation knowledge. (X/Y): X = number of queries that Clique reached the accuracy; Y = total query count.	113
5.11	The average delay to reach 0.99 under different pre-processing budgets during ingestion. (X/Y): X = starter cameras covered; Y = total number of locations. X > Y means covering more than one starter cameras per location.	114

ABSTRACT

Pervasive cameras are generating videos at an unprecedented pace, making videos the new frontier of big data. As the processors, e.g., CPU/GPU, become increasingly powerful, the cloud and edge nodes can generate useful insights from colossal video data. However, as the research in computer vision (CV) develops vigorously, the system area has been a blind spot in CV research. With colossal video data generated from cameras every day and limited compute resource budgets, how to design software systems to generate insights from video data efficiently?

Designing cost-efficient video analytics software systems is challenged by the expensive computation of vision operators, the colossal data volume, and the precious wireless bandwidth of surveillance cameras. To address above challenges, three software systems are proposed in this thesis. For the first system, we present VStore, a data store that supports fast, resource-efficient analytics over large archival videos. VStore manages video ingestion, storage, retrieval, and consumption and controls video formats through backward derivation of configuration: in the opposite direction along the video data path, VStore passes the video quantity and quality expected by analytics backward to retrieval, to storage, and to ingestion. VStore derives an optimal set of video formats, optimizes for different resources in a progressive manner, and runs queries as fast as $362\times$ of video realtime. For the second system, we present a camera/cloud runtime called DIVA that supports querying cold videos distributed on low-cost wireless cameras. DIVA is built upon a novel zero-streaming paradigm: to save wireless bandwidth, when capturing video frames, a camera builds sparse yet accurate landmark frames without uploading any video data; when executing a query, a camera processes frames in multiple passes with increasingly more expensive operators. On diverse queries over 15 videos, DIVA runs at more than $100\times$ realtime and outperforms competitive alternatives remarkably. For the third system, we present Clique, a practical object re-identification (ReID) engine that builds upon two unconventional techniques. First, Clique assesses target occurrences by clustering unreliable object features extracted by ReID algorithms, with each cluster representing the general impression of a distinct object to be matched against the input. Second, to search across camera videos, Clique samples cameras

to maximize the spatiotemporal coverage and incrementally adds cameras for processing on demand. Through evaluation on 25 hours of traffic videos from 25 cameras, Clique reaches a high recall at 5 of 0.87 across 70 queries and runs at 830 \times of video realtime in achieving high accuracy.

1. INTRODUCTION

Video has become the new frontier for big data as pervasive cameras produce videos every day. Over the past 10 years, the annual shipments of surveillance cameras grow by $10\times$, to 130M per year [1]. Many campuses are reported to run more than 200 cameras 24×7 [2]. A survey of 61 organizations shows that from 2015 to 2018, the average number of cameras has increased by almost 70%, from 2,900 to 4,900 [3]. In such deployment, a single camera produces as much as tens of GB encoded video footage per day (720P at 30 fps). Such volume of video footage is infeasible for a human to consume who typically watches videos at 30-60 fps.

Offloading video consumption to machines is a promising solution. With the advance of deep learning [4] since 2015, deep neural networks (DNNs) have become a promising video consumer, offering a wide spectrum of vision operators with different functions under various consumption speeds. By fully utilizing DNNs, machines are able to generate useful insights from video footage captured by surveillance cameras. For example, to trace the cause of recent frequent congestion on a high way, a city planner queries cameras on nearby local roads, requesting car counts seen on these local roads; to investigate a crime happened last night, a police officer queries corresponding camera video footage to identify any human in blue shirts. To speed up video queries, a video query is typically executed as a cascade of vision operators [5]–[9], which enables answering video queries from hour-long videos in seconds.

Video analytics is expensive. A \$4000 GPU is often required to generate useful insights from the videos. State-of-the-art object detectors, e.g., YOLOv3 [10] only runs at 20 fps. On the contrary, storage devices are cheap, e.g., a 5-TB hard drive that costs only \$200 can store a few months of videos. Retrospective analytics, which stores the the video data to the disks and only retrieve them on demand when the user queries, is more cost-efficient.

Retrospective analytics offers several key advantages that live analytics lacks. (i) Analyzing many video streams in real time is expensive, e.g., running DNNs over live videos 24×7 . It also requires a deployment of always-on network configuration. (ii) Query types may only become known after the video capture [7]. (iii) At query time, users may interactively revise

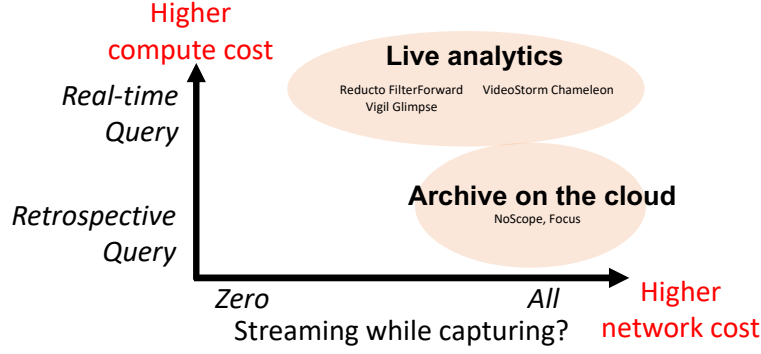


Figure 1.1. The design space of existing video software systems [6], [8], [9], [11]–[15].

their query types or parameters [16], which may not be foreseen at ingestion time. (iv) In many applications, only a small fraction of the video will be eventually queried [17], making live analytics an overkill.

As shown in Figure 1.1, most existing video systems focus on real-time video analytics. This thesis explores the retrospective video analytics and adds new points to the design space. The remainder of this thesis is structured as follows: in Chapter 3, we will present VStore¹ [18], a data store that supports fast, resource-efficient video analytics, which manages video ingestion, storage, retrieval, and consumption through automatic configuration of video formats; in Chapter 4, we will present DIVA² [19], a query engine that supports querying low-cost wireless cameras with limited compute and network resources; in Chapter 5, we will present Clique [20], an object ReID query engine that supports efficient spatiotemporal queries on city-scale camera deployments.

Ethical considerations In this thesis, all visual data used is from the public domain. No information traceable to human individuals is collected or analyzed.

1.1 Supporting video storage for cost-efficient retrospective video analytics

As recent query engines [7], [8] assume *all* input data as raw frames present in memory, there lacks a video store that manages large videos for analytics. The data store should orchestrate four major stages on the video data path: ingestion, storage, retrieval, and

¹↑A version of this work was previously published in Proceedings of the Fourteenth EuroSys Conference, 2019 (EuroSys ’19). <https://doi.org/10.1145/3302424.3303971>

²↑A version of this work is pending publication in USENIX Annual Technical Conference, 2021 (ATC ’21)

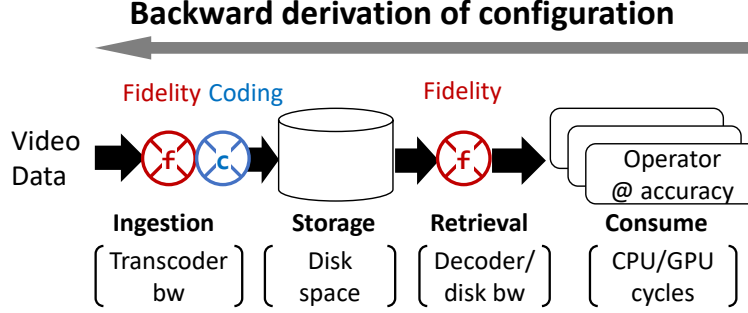


Figure 1.2. The VStore architecture, showing the video data path and backward derivation of configuration.

consumption, as shown in Figure 1.2. The four stages demand multiple hardware resources, including encoder/decoder bandwidth, disk space, and CPU/GPU cycles for query execution. The resource demands are high, thanks to large video data. Demands for different resource types may conflict. Towards optimizing these stages for resource efficiency, classic video databases are inadequate [21]: they were designed for *human* consumers watching videos at $1\times$ – $2\times$ speed of video realtime; they are incapable of serving some *algorithmic* consumers, i.e., operators, processing videos at more than $1000\times$ video realtime. Shifting part of the query to ingestion [15] has important limitations and does not obviate the need for such a video store.

Towards designing a video store, we advocate for taking a key opportunity: as video flows through its data path, the store should control video formats (fidelity and coding) through extensive video parameters called *knobs*. These knobs have significant impacts on resource costs and analytics accuracy, opening a rich space of trade-offs.

We present VStore, a system managing large videos for retrospective analytics. The primary feature of VStore is its automatic configuration of video formats. As video streams arrive, VStore saves multiple video versions and judiciously sets their *storage formats*; in response to queries, VStore retrieves stored video versions and converts them into *consumption formats* catering to the executed operators. Through configuring video formats, VStore ensures operators to meet their desired accuracies at high speed; it prevents video retrieval from bottlenecking consumption; it ensures resource consumption to respect budgets.

To decide video formats, VStore is challenged by i) an enormous combinatorial space of video knobs; ii) complex impacts of these knobs and high profiling costs; iii) optimizing for multiple resource types. These challenges were unaddressed: while classic video databases may save video contents in multiple formats, their format choices are oblivious to analytics and often ad hoc [21]; while existing query engines recognize the significance of video formats [6], [7], [9] and optimize them for query execution, they omit video coding, storage, and retrieval, which are all crucial to retrospective analytics.

To address these challenges, our key idea behind VStore is *backward derivation*, shown in Figure 1.2. In the opposite direction of the video data path, VStore passes the desired data quantity and quality from algorithmic consumers backward to retrieval, to storage, and to ingestion. In this process, VStore optimizes for different resources in a progressive manner; it elastically trades off among them to respect resource budgets. More specifically, i) from operators and their desired accuracies, VStore derives video formats for fastest data consumption, for which it effectively searches in a high-dimensional parameter space with video-specific heuristics; ii) from the consumption formats, VStore derives video formats for storage, for which it systematically coalesces video formats to optimize for ingestion and storage costs; iii) from the storage formats, VStore derives a data erosion plan, which gradually deletes aging video data, trading off analytics speed for lower storage cost.

Through evaluation with two real-world queries over six video datasets, we demonstrate that VStore is capable of deriving large, complex configuration with hundreds of knobs over tens of video formats, which are infeasible for humans to tune. Following the configuration, VStore stores multiple formats for each video footage. To serve queries, it streams video data (encoded or raw) from disks through decoder to operators, running queries as fast as $362\times$ of video realtime. As users lower the target query accuracy, VStore elastically scales down the costs by switching operators to cheaper video formats, accelerating the query by two orders of magnitude. This query speed is $150\times$ higher compared to systems that lack automatic configuration of video formats. VStore reduces the total configuration overhead by $5\times$.

We have made the following contributions in VStore.

- We make a case for a new video store for serving retrospective analytics over large videos. We formulate the design problem and experimentally explore the design space.
- To design such a video store, we identify the configuration of video formats as the central concern. We present a novel approach called backward derivation. With this approach, we contribute new techniques for searching large spaces of video knobs, for coalescing stored video formats, and for eroding aging video data.
- We report VStore, a concrete implementation of our design. Our evaluation shows promising results. VStore is the first holistic system that manages the full video lifecycle optimized for retrospective analytics, to our knowledge.

1.2 Supporting exploratory video queries on low-cost wireless cameras

Four recent trends motivate DIVA.

(1) *Low-cost, wireless cameras grow fast.* As key complements to high-end cameras, low-cost cameras (<\$40) are increasingly pervasive [22]–[24]. These cameras often have limited compute resources yet spacious storage. Being wireless, these cameras are meant to be installed by individuals or small businesses with ease just as other wireless sensors.

(2) *Most videos are cold.* Users typically deploy cameras for capturing *excessive* videos, despite knowing most videos will never be queried. This is because interesting events are often unforeseeable, e.g., car accidents; the need for examining such events emerges well after the fact. Section 4.1.1 presents a 6-month study of real-world camera deployment, where only <0.005% of captured videos is eventually queried.

(3) *Transmitting cold videos wastes wireless bandwidth.* Cold videos should never contend with human users for network bandwidth. If streaming video in real-time, a *single* camera generates traffic at 0.2–0.4 MB/s (720P at 1–30 FPS); with *multiple* cameras on one network, their always-on streams easily consume most, if not all, bandwidth of consumer WiFi, which is 0.2–3 MB/s (median: 0.99) in a recent global survey [25] and less than 1.5 MB/s in an academic study [26]. A dedicated network for cameras is expensive, as network monetary cost exceeds camera cost in three months [27].

(4) *Camera storage can retain videos long enough.* A cheap camera can already store videos for weeks or months. Such retention periods already satisfy many video scenarios [28], [29]. In fact, legal regulations often *prevent* retention longer than a few months, mandating video deletion for privacy [30], [31]. Existing security measures can assure security of on-camera videos. Section 4.1.3 will provide evidence in detail.

Zero streaming How to analyze cold videos produced by many low-cost cameras? We advocate for a system model dubbed “zero streaming”. (1) Cameras continuously capture videos to their local storage without uploading any. (2) Only in response to a retrospective query, the cloud reaches out to the queried camera, coordinating with it to process the queried video. (3) While the video is being processed, the system presents users with inexact yet useful results; it continuously refines the results until query completion [32]. In this way, a user may *explore* videos through interactive queries, e.g., aborting an ongoing query based on inexact results and issuing a new query with revised parameters [16], [33]. Zero streaming has rich use cases. For example, to understand how recent visitors impact bobcat activities, a ranger queries all the park’s cameras, requesting time ranges where the cameras capture bobcats.

Zero streaming suits resource-frugal cameras in large deployment. When capturing videos, cameras require no network or external compute resources. Only to process a query, the cameras require network, e.g., long-range wireless [34], and cloud compute resources, e.g., GPU. Zero streaming adds a new point to the design space of video analytics shown in Figure 1.1. It facilitates retrospective, exploratory analytics, a key complement to real-time event detection and low-delay video retrieval [6], [9], [11], [15]. The latter demands higher compute or network resources per camera and hence suits fewer cameras on hot locations, e.g., building entrances.

DIVA To support querying zero-streaming cameras, we present a camera/cloud runtime called DIVA. As shown in Figure 1.3, a camera captures video to local storage; it deletes videos after their maximum retention period. In response to a query, the camera works in conjunction with the cloud: the camera runs operators, implemented as lightweight neural nets (NNs), to *rank* or *filter* frames; the cloud runs full-fledged object detection to validate

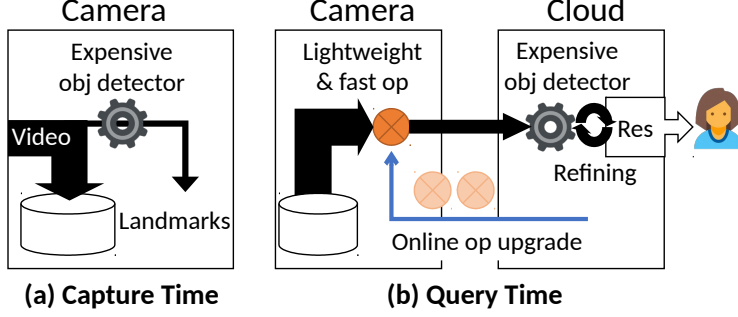


Figure 1.3. Overview of DIVA.

results uploaded from the camera. DIVA thus does not sacrifice query accuracy, ensuring it as high as that of object detection by the cloud.

The major challenges to DIVA are two. (1) During video capture: how should cameras best use limited resources for future queries? (2) To execute a query: how should the cloud and the camera orchestrate to deliver useful results rapidly? Existing techniques are inadequate. Recent systems pre-process (“index”) video frames as capturing them [15] and answer queries based on indexes only. Yet, as we will show in the Section 4.7, low-cost cameras can hardly build quality indexes in real-time. Many systems process video frames in a streaming fashion [12]–[14], [35], [36], which however miss key opportunities in retrospective queries.

To this end, DIVA has two unconventional designs.

- **During video capture: building sparse but sure landmarks to distill long-term knowledge** (Figure 1.3(a)) To optimize future queries, our key insight is that *accurate* knowledge on a *sparse* sample of frames is much more useful than *inaccurate* knowledge on *all* frames. This is opposite to existing designs that detect objects with low accuracy on all/most frames as capturing them [12], [15]. On a small sample of captured video frames dubbed *landmarks*, the camera runs generic, expensive object detection, e.g., the YOLOv3 [10]. Constrained by the camera hardware, landmarks are sparse in time, e.g., 1 in every 30 seconds; yet, with high-accuracy object labels, they provide *reliable* spatial distributions of various objects over long-term videos. High accuracy is crucial, as we will validate through evaluation in Section 4.7.2. DIVA

uses landmarks for optimizing queries: it prioritizes processing of frame regions with object skewness learned from landmarks; it bootstraps operators with landmarks as training samples. Landmarks only capture a small fraction of object instances; those uncaptured do not affect correctness/accuracy (Section 4.3).

- **To execute queries: multipass processing with online operator upgrade** (Figure 1.3(b)) To process large videos, our key insight is to refine query results in multiple passes, each pass with a more expensive/accurate operator. Unlike prior systems processing all frames in one pass and delivering results in one shot [7], [8], [12], multipass processing produce useful results during query execution, enabling users to explore videos effectively. To do so, DIVA’s cloud trains operators with diverse accuracy/costs. Throughout query execution, the cloud keeps pushing new operators to the camera, picking the next operator based on query progress, network conditions, and operator accuracy. The early operators quickly *explore* the frames for inexact answers while later operators slowly *exploit* for more exact answers.

On 720-hour videos in total from 15 different scenes, DIVA runs queries at more than $100\times$ video realtime on average, with typical wireless conditions and low-cost hardware. DIVA returns results quickly: compared to executing a query to completion, DIVA takes one order of magnitude shorter time to return half of the result frames. Compared to competitive alternatives, DIVA speeds up queries by at least $4\times$.

We have made the following contributions in DIVA.

- Zero streaming, a new model for low-cost cameras to operate on frugal networks while answering video queries.
- Two novel techniques for querying zero-streaming cameras: optimizing queries with accurate knowledge from sparse frame samples rather than inaccurate knowledge on all frames; processing frames in multiple passes with operators continuously picked during a query, rather than one-pass processing with operators decided ahead of a query.

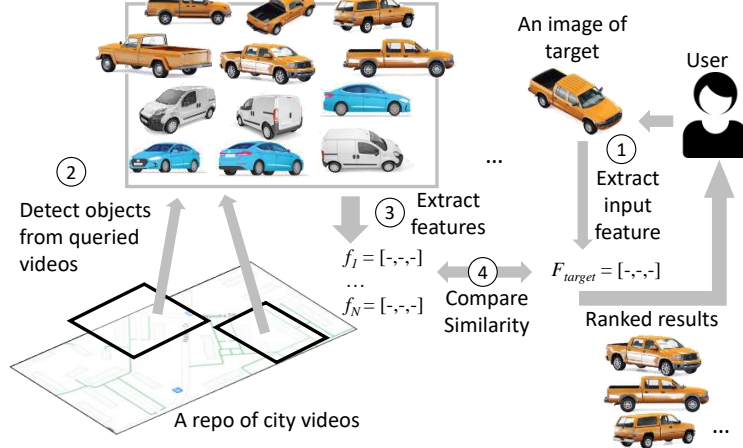


Figure 1.4. The classic pipeline for object ReID, formulated as image retrieval.

- DIVA, a concrete implementation that runs queries at more than 100× realtime with uncompromised query accuracy.

1.3 Supporting object re-identification under city-scale camera deployments

City-scale camera deployment As video intelligence advances and camera cost drops, city cameras expand fast. Strategically deployed near key locations, such as highway entrances or road intersections, multiple cameras (reported to be 2–5 per location [37], [38]) offer complementary, often overlapped viewpoints of scenes.

Object ReID on city videos A key application of city cameras is object re-identification (ReID): given an input image of an object X , searching for occurrences of X in a video repository. ReID has been an important computer vision task, seeing popular use cases such as crime investigation and traffic planning [38]–[40]. Many ReID algorithms are proposed recently, fueled by neural networks [41]–[48]. Object ReID over city videos is typically “finding a needle in haystack”. The queried videos are long and produced by many cameras; the videos may not contain the input image, or any images from the camera that produced the input image (called the *origin* camera); the occurrences of target object can be rare and transient. For instance, in a popular dataset of city traffic videos [37], 99% of vehicles only appear for less than 10 seconds.

The common pipeline structure for ReID is shown in Figure 1.4: ① given an input image of target object X, the pipeline extracts its feature, e.g., using ResNet-152 [49] to extract a 1024-dimension vector [46], [47]; ② from the queried videos, the pipeline detects all bounding boxes belonging to the same *class* as X, e.g., using YOLO [50]; ③ the pipeline extracts features of all detected bounding boxes; ④ it calculates pairwise similarities between X and the bounding boxes. The similarity is often measured as feature distance [51], where a shorter distance suggests a higher similarity between X and a bounding box. Of the four stages, stage 2 and 3 are most expensive. For instance, calculating feature distances in stage 4 is three orders of magnitude faster than extracting the features in stage 3. The cost of stage 2 and 3 further grows with the amount of videos. This pipeline structure is widely used, e.g., by almost all participants in popular vehicle ReID challenges [46], [47], [52].

Proliferating ReID algorithms call for a practical ReID system. Our driving use case is vehicle ReID, where personal identifiable information such as license plates are intentionally removed for privacy [37]. Vehicle ReID is considered one of the most important ReID problems [40]. The techniques are likely transferable to other object classes.

To design a robust and efficient system for ReID, we have to address two challenges. First, modern ReID algorithms are not always reliable. By its definition in computer vision, ReID focuses on differentiating numerous objects of the *same class*, e.g., cars. In real-world videos, however, many objects of the same class exhibit minor visual differences; yet bounding boxes of the same object – captured by the same or different cameras – may appear quite different. As we will show in Section 5.1, even sophisticated feature extractors may deem bounding boxes of *different* objects more similar than bounding boxes of the *same* object. Second, the number of cameras and the volume of videos are colossal. With 2-5 cameras per intersection [37], [38] and 60–500 intersections per square mile in urban areas [53], a query covering a few square miles would need to process a few hundred, if not a few thousand, cameras. Furthermore, modern ReID pipelines have an insatiable need for resources. For example, Titan V, a \sim \\$3,000 modern GPU, runs YOLO [54], [55] for detecting bounding boxes at only 40 FPS. The GPU running ResNet-152 extracts \sim 80 features per second. To process city videos from one square mile in a day, we estimate at least several hundred GPU hours are needed. This cost quickly becomes prohibitive as camera deployment and

query scope expands. Resorting to cheap vision algorithms, e.g., smaller neural networks or SIFT, is unlikely to help: they are much more susceptible to subtle visual differences and disturbance, making ReID results even less usable.

Principles While prior research formulates ReID as image retrieval queries, i.e., to find every bounding box of a target object X [46], [47], [56], [57], we treat ReID as *spatiotemporal* queries, which search for what users care about: the times and locations in which object X appeared. This gives opportunities to overcome the accuracy limitation on individual bounding boxes, and to quickly emit times and locations before processing all bounding boxes in the queried videos. We address the multitude of city cameras by renewing a wisdom in video analytics: resource/quality tradeoffs [6], [8], [9], [15], [18]. Prior video systems often target fewer cameras, making such tradeoffs *within* a video stream, e.g., by tuning frame resolutions, rates, and cropping factors. On city-scale videos, however, processing more cameras is almost always favorable than processing more pixels from each camera. This is because: (1) cameras in different locations provide extensive spatial coverage; (2) cameras in the same location provide complementary viewpoints. Both factors benefit ReID queries more than video quality. To this end, we prioritize increasing camera coverage over increasing video quality, e.g., frame rate and resolution.

We make minimum, qualitative assumptions on camera deployment. Quantitative deployment knowledge, e.g., camera orientations and correlations, used to enable optimizations within smaller camera networks [56]. For emerging city-scale cameras, however, it is unclear if there exists a generic, quantitative deployment model. Minimizing assumptions allow a generic system design, which, as we will demonstrate, serves as the basis for deployment-specific optimizations.

Clique We present a ReID engine called Clique. Catering to spatiotemporal queries, Clique organizes all videos in a repository as spatiotemporal cells, where a cell $\langle L, T \rangle$ contains video clips captured by all cameras near a geo-location L during a time period T . Clique answers a query for target object X with a short list of spatiotemporal cells, ranked by their promises of containing X ; each returned cell is accompanied by video clips, with annotations of the likely bounding boxes of X . As executing a query, Clique keeps updating

the rank based on new results from video processing. The user reviews returned cells and makes the final decision.

We design and evaluate Clique as a recall-oriented system [58]: it seeks to find all positive cells (which are rare) and rank them to the top. As such, Clique minimizes human efforts in analyzing videos; it does not seek to replace humans, whose knowledge cannot (yet) be substituted by algorithms on real-world videos. This goal is shared by existing recall-oriented systems, e.g., for legal documents or patents search [59], [60], where final decisions from humans are indispensable.

Clique have two key designs. First, we clusters unreliable bounding boxes to approximate distinct objects from sparse videos. Our insight is: how is an object perceived by a camera is heavily impacted by (1) the camera’s posture, including position and orientation; (2) transient disturbance, such as occlusion and background clutter. These impacts sometimes overshadow the object’s characteristics, e.g., shape and color. To counter the two impacts, Clique *matches* the origin camera’s posture: it samples diverse co-located cameras in hope of finding ones with postures similar to the input. Clique *mitigates* the disturbance: it clusters similar bounding boxes captured by a camera during a period of time. Each resultant cluster thus represents the camera’s general “impression” of a distinct object. Clique estimates the occurrence of X based on the similarity between the input image and distinct objects as represented by clusters. Clustering has been a classic algorithm in data processing [61]–[63] especially in vision [15]; Clique is novel in applying it to ReID, deriving robust query answers from unreliable bounding box features. Clustering suits our principle of prioritizing camera coverage, as it tolerates low frame rate on each camera. Second, Clique searches incrementally in spatiotemporal cells. The key is the camera sampling strategy: to avoid redundant video contents as much as possible while exploiting diverse camera postures as needed. Clique navigates its resource spending towards cells where new discovery of target occurrences is most likely. To do this, Clique starts a query with a minimum number of cameras to quickly estimate promises of all cells; it processes videos from additional cameras for undecided cells; it iteratively assesses cell promises and re-ranks all cells for subsequent search.

We implement Clique and evaluate it on a video dataset of 25 hours of videos from 25 cameras. On 70 queries with different target objects, Clique on average delivers a high recall at 5 of 0.87; it reaches high an accuracy goal of 0.99 in 108.5 seconds on average, at the speed of $830\times$ of video realtime. Compared to alternative designs, Clique reduces query delays by up to $6.5\times$. We further evaluate deployment-specific optimizations.

We made the following contributions in Clique.

- Towards a practical ReID system, we advocate a new approach: focusing on finding relevant spatiotemporal cells rather than individual object instances.
- We present to cluster unreliable bounding boxes as approximations of distinct objects, which effectively overcomes limitation in ReID algorithm accuracy.
- We present incremental search in cells. This minimizes redundant processing while exploiting diverse camera viewpoints, which reduces the ReID compute cost.
- We report Clique, a ReID system that works on large video repositories.

2. SUMMARY

This thesis presents three software systems, i.e., VStore, DIVA, and Clique, that support efficient large-scale retrospective video analytics.

The primary feature of VStore is its automatic configuration of video formats. As video streams arrive, VStore saves multiple video versions and judiciously sets their *storage formats*; in response to queries, VStore retrieves stored video versions and converts them into *consumption formats* catering to the executed operators. Through configuring video formats, VStore ensures operators to meet their desired accuracy levels at high speed; it prevents video retrieval from bottlenecking consumption; it ensures resource consumption to respect budgets. To decide video formats, VStore is challenged by an enormous combinatorial space of video knobs, complex impacts of these knobs and high profiling costs, and the optimization for multiple resource types. The key idea behind VStore is *backward derivation*: in the opposite direction of the video data path, VStore passes the desired data quantity and quality from algorithmic consumers backward to retrieval, to storage, and to ingestion. In this process, VStore optimizes for different resources in a progressive manner; it elastically trades off among them to respect resource budgets. VStore runs queries as fast as 362× of video realtime.

DIVA is an analytics engine for querying cold videos on remote low-cost wireless cameras. It is built upon a novel system model called “zero streaming” that shifts most compute from capture time to query time. At capture time, DIVA builds sparse but sure landmarks. At query time, the camera works in conjunction with the cloud: the camera runs operators, implemented as lightweight neural nets (NNs), to *rank* or *filter* frames; the cloud runs full-fledged object detection to validate results uploaded from the camera. DIVA thus does not sacrifice query accuracy, ensuring it as high as that of object detection by the cloud. In this way, a user may *explore* videos through interactive queries, e.g., aborting an ongoing query based on inexact results and issuing a new query with revised parameters [16], [33]. Our evaluation of three types of queries shows that DIVA can run at more than 100× video realtime under typical wireless network and camera hardware.

Clique is a practical object ReID engine that answers spatiotemporal queries. Clique organizes all videos in a repository as spatiotemporal cells, where a cell $\langle L, T \rangle$ contains video clips captured by all cameras near a geo-location L during a time period T . Clique answers a query for target object X with a short list of spatiotemporal cells, ranked by their promises of containing X ; each returned cell is accompanied by video clips, with annotations of the likely bounding boxes of X . When executing a query, Clique keeps updating the rank based on new results from video processing. The user reviews returned cells and makes the final decision. Clique is built upon two unconventional designs. First, Clique approximates each distinct object by clustering fuzzy object features emitted by ReID algorithms before matching with the input image containing the target object. Second, to search in colossal video data, Clique samples cameras to maximize the spatiotemporal coverage and incrementally searches in additional cameras on demand. On 25 hours of city videos spanning 25 cameras, Clique on average reached an recall at 5 of 0.87 and runs at $830\times$ video real time in achieving high accuracy.

3. VSTORE: A DATA STORE FOR ANALYTICS ON LARGE VIDEOS

3.1 Background & motivations

3.1.1 Retrospective video analytics

Query & operators A video query is typically executed as a cascade of operators. As shown in Figure 3.1, early operators scan most of the queried video time span at low cost. They activate late operators over a small fraction of video for deeper analysis. Operators consume raw video frames. Of a cascade, the execution costs of operators can differ by three orders of magnitude [8]; they also prefer different input video formats, catering to their internal algorithms.

Accuracy/cost trade-offs in operators An operator’s output quality is characterized by *accuracy*, i.e., how close the output is to the ground truth. We use a popular accuracy metric called F1 score: the harmonic mean of precision and recall [6]. At runtime, an operator’s target accuracy is set in queries [6], [7], [9]. VStore seeks to provision minimum resources for operators to achieve the target accuracy.

3.1.2 System model

We consider a video store running on one or a few commodity servers. Incoming video data flows through the following major system components. We assume a pre-defined library of operators, the number of which can be substantial; each operator may run at a pre-defined set of accuracy levels. By combining the existing operators at different accuracy levels, a variety of queries can be assembled. We will discuss how operator addition/deletion may be handled in Section 3.6.

- **Ingestion:** Video streams continuously arrive. In this work, we consider the input rate of incoming video as given. The ingestion optionally converts the video formats, e.g., by resizing frames. It saves the ingested videos either as encoded videos (through transcoding) or as raw frames. The ingestion throughput is bound by transcoding

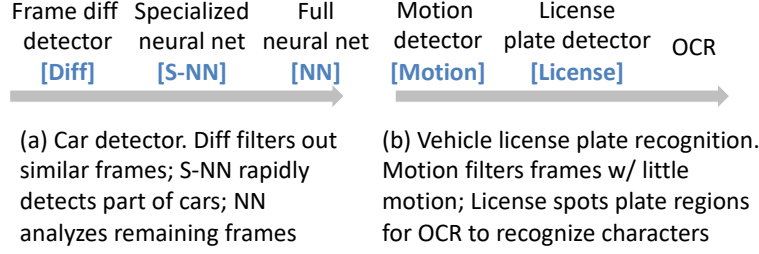


Figure 3.1. Video queries as operator cascades [8], [64].

bandwidth, typically one order of magnitude lower than disk bandwidth. This paper will present more experimental results on ingestion.

- **Storage:** Like other time-series data stores [65], videos have age-based values. A store typically holds video footage for a user-defined lifespan [66]. In queries, users often show higher interest in more recent videos.
- **Retrieval:** In response to operator execution, the store retrieves video data from disks, optionally converts the data format for the operators, and supplies the resultant frames.

If the on-disk videos are encoded, the store must decode them before supplying. Data retrieval may be bound by decoding or disk read speed. Since the decoding throughput (often tens of MB/sec) is far below disk throughput (at least hundreds of MB/sec), the disk only becomes the bottleneck in loading *raw* frames.
- **Consumption:** The store supplies video data to consumers, i.e., operators spending GPU/CPU cycles to consume data.

Figure 1.2 summarizes the resource cost of the components above. The retrieval/consumption *costs* are reciprocal to data retrieval/consumption *speed*, respectively. The operator runs at the speed of retrieval or consumption, whichever is lower. To quantify operator speed, we adopt as the metric the ratio between video duration and video processing delay. For instance, if a 1-second video is processed in 1 ms, the speed is 1000× realtime.

Key opportunity: controlling video formats As video data flows through, a video store is at liberty to control the video formats. This is shown in Figure 1.2. At the ingestion,

the system decides *fidelity* and *coding* for each stored video version; at the data retrieval, the system decides the *fidelity* for each raw frame sequence supplied to consumers.

Running operators at ingestion is not a panacea. Recent work runs early-stage operators at ingestion to save executions of expensive operators at query time [15]. This approach has important limitations.

- It bakes query types in the ingestion. Video queries and operators are increasingly rich [67]–[71]; one operator (e.g., neural networks) may be instantiated with different parameters depending on training data [33]. Running all possible early operators at ingestion is therefore expensive.
- It bakes specific accuracy/cost trade-offs in the ingestion. Yet, users at query time often know better trade-offs, based on domain knowledge and interactive exploration [7], [16].
- It prepays computation cost for all ingested videos. In many scenarios such as surveillance, only a small fraction of ingested video is eventually queried [2], [33]. As a result, most operator execution at ingestion is in vain.

In comparison, by preparing data for queries, a video store supports richer query types, incurs lower ingestion cost, and allows flexible query-time trade-offs. Section 3.6 will provide further discussion.

3.1.3 Video format knobs

The video format is controlled by a set of parameters, or knobs. Table 3.1 summarizes the knobs considered in this work, chosen due to their high resource impacts.

Fidelity knobs For video data, encoded or raw, fidelity knobs dictate i) the *quantity* of visual information, e.g., frame sampling which decides the frame rate; ii) the *quality* of visual information, which is subject to the loss due to video compression. Each fidelity knob has a finite set of possible values. A combination of knob values constitutes a *fidelity option* f . All possible fidelity options constitute a fidelity space \mathbb{F} .

“Richer-than” order Among all possible values of one fidelity knob, one may establish a *richer-than* order (e.g., 720p is richer than 180p). Among fidelity *options*, one may establish

Table 3.1. Knobs and their values considered in this work. Total: 7 knobs and 15K possible combinations of values. Note: no video quality and coding knobs for RAW.

<i>Fidelity knob</i>	Values	<i>Coding knob</i>	Values
Img. quality	worst, bad, good, best *	Speed step	slowest, slow, med, fast, fastest**
Crop factor	50%, 75%, 100%	KFrame int.	5,10,50,100,250
Resolution	60x60 ... 720p (total 10)	Bypass	Y or N (Y=raw)
Fr. sampling	1/30, 1/5, 1/2, 2/3, 1		

Equivalent FFmpeg options:

* CRF = 50, 40, 23, 0

**preset = veryslow, medium, veryfast, superfast, ultrafast

a partial order of *richer-than*: option X is *richer than* option Y if and only if X has the same or richer values on all knobs and richer values on at least one knob. The *richer-than* order does *not* exist in all pairs of fidelity options, e.g., between good-50%-720p-1/2 and bad-100%-540p-1. One can degrade fidelity X to get fidelity Y only if X is richer than Y.

Coding Knobs Coding reduces raw video size by up to two orders of magnitude [72]. Coding knobs control encoding/decoding speed and the encoded video size. Orthogonal to video fidelity, coding knobs provide valuable trade-offs among the costs of ingestion, storage, and retrieval. These trade-offs do not affect consumer behaviors – an operator’s accuracy and consumption cost.

While a modern encoder may expose tens of coding knobs (e.g., around 50 for x264), we pick three for their high impacts and ease of interpretation. Table 3.1 summarizes these knobs and Figure 3.2 shows their impacts. **Speed step** accelerates encoding/decoding at the expense of increased video size. As shown in Figure 3.2(a), it can lead up to 40× difference in encoding speed and up to 2.5× difference in storage space. **Keyframe interval**: An encoded video stream is a sequence of chunks (also called “group of pictures” [73]): beginning with a key frame, a chunk is the smallest data unit that can be decoded independently. The keyframe interval offers the opportunity to accelerate decoding if the consumers only sample to consume a fraction of frames. If the frame sampling interval N is larger than the keyframe interval M , the decoder can skip N/M chunks between two adjacent sampled frames without decoding these chunks. In the example in Figure 3.2(b), smaller keyframe intervals increase

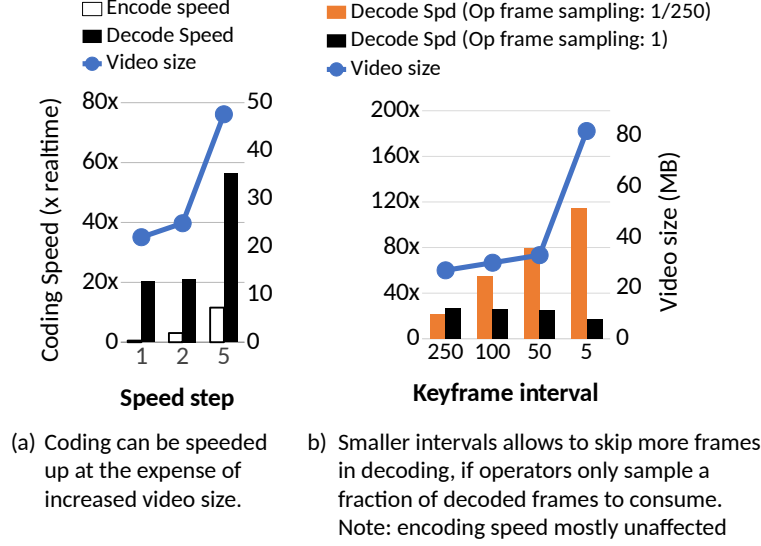


Figure 3.2. Impacts of coding knobs. Video: 100 seconds from *tucson*. See Section 3.5 for dataset and test hardware.

decoding speed by up to $6\times$ at the expense of larger encoded videos. **Coding bypass:** The ingestion may save incoming videos as raw frames on disks. The resultant extremely low retrieval cost is desirable to some fast consumers.

A combination of coding knob values is a coding option c . All possible coding options constitute a coding space \mathbb{C} .

3.1.4 Knob impacts

As illustrated in Figure 1.2: for on-disk videos, fidelity and coding knobs jointly impact the costs of ingestion, storage, and retrieval; for in-memory videos to be consumed by operators, fidelity knobs impact the consumption cost and the consuming operator’s accuracy. We have a few observations. First, fidelity knobs enable rich cost/accuracy trade-offs. As shown in Figure 3.3, one may reduce resource costs by up to 50% with minor (5%) accuracy loss. Second, the knobs enable rich trade-offs among resource types. This is exemplified in Figure 3.4: although three video fidelity options all lead to similar operator accuracy (0.8), there is no single most resource-efficient one, e.g., fidelity B incurs the lowest *consumption* cost, but the high *storage* cost due to its high image quality. Third, each knob has significant impacts. Take Figure 3.3(b) as an example: one step change to image quality reduces

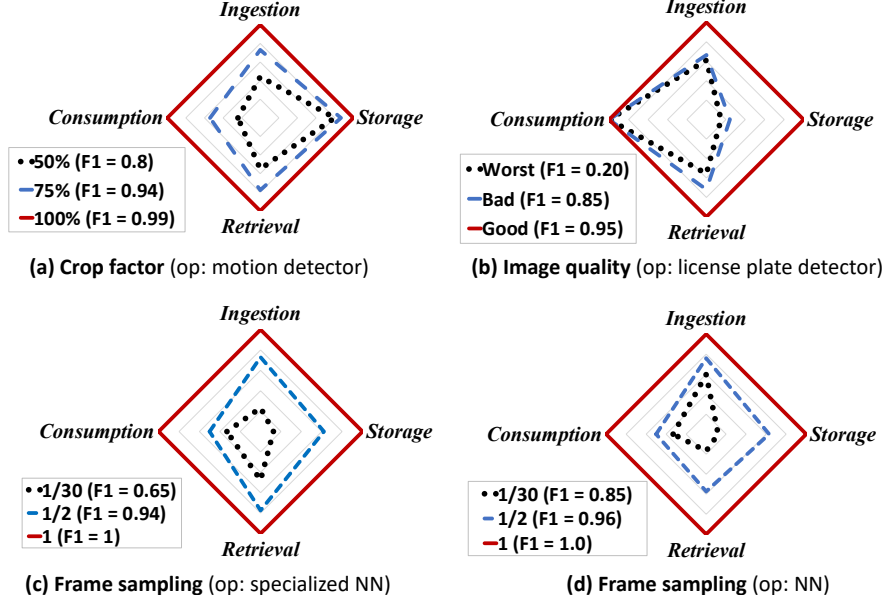


Figure 3.3. Fidelity knobs have high, complex impacts on costs of multiple components (normalized on each axis) and operator accuracy (annotated in legends). Each plot: one knob changing; all others fixed. See Section 3.5 for methodology.

accuracy from 0.95 to 0.85, the storage cost by $5\times$, and the ingestion cost by 40%. Fourth, omitting knobs misses valuable trade-offs. For instance, to achieve high accuracy of 0.9, the license detector would incur 60% more consumption cost when the image quality of its input video changes from “good” to “bad”. This is because the operator must consume higher *quantity* of data to compensate for the lower *quality*. Yet, storing all videos with “good” quality requires $5\times$ storage space. Unfortunately, most prior video analytics systems fix the image quality knob at the default value.

The quantitative impacts are complex. i) The knob/cost relations are difficult to capture in analytical models [9]. ii) The quantitative relations vary across operators and across video contents [6]. This is exemplified by Figure 3.3 (c) and (d) that show the same knob’s different impacts on two operators. iii) One knob’s impact depends on the values of other knobs. Take the license detector as an example: as image quality worsens, the operator’s accuracy becomes more sensitive to resolution changes. With “good” image quality, lowering image resolution from 720p to 540p slightly reduces the accuracy, from 0.83 to 0.81; with “bad” image quality, the same resolution reduction significantly reduces the accuracy, from

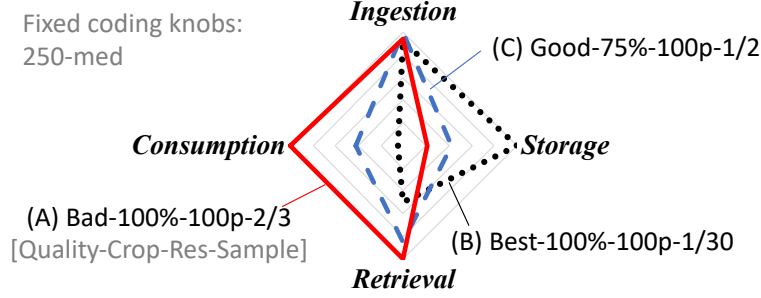


Figure 3.4. Disparate costs of fidelity options A–C, despite all leading to operator accuracy ≈ 0.8 . Operator: License. Cost normalized on each axis. See Section 3.5 for methodology.

0.76 to 0.52. While prior work assumes that certain knobs have independent impacts on accuracy [6], our observation shows that dependency exists among a larger set of knobs.

Summary & discussion Controlling video formats is central to a video store design. The store should actively manage fidelity and coding throughout the video data path. To characterize knob impacts, the store needs regular profiling. Some video analytics systems recognize the significance of video formats [6], [7], [9]. However, they focus on optimizing query execution yet omitting other resources, such as storage, which is critical to retrospective analytics. They are mostly limited to only two fidelity knobs (resolution and sampling rate) while omitting others, especially coding. As we will show, a synergy between fidelity and coding knobs is vital.

3.2 A case for a new video store

We set to design a video store that automatically creates and manages video formats in order to satisfy algorithmic video consumers with high resource efficiency.

3.2.1 The configuration problem

The store must determine a global set of video formats as follows.

- **Storage format:** the system may save one ingested stream in multiple versions, each characterized by a fidelity option f and a coding option c . We refer to $SF\langle f, c \rangle$ as a storage format.

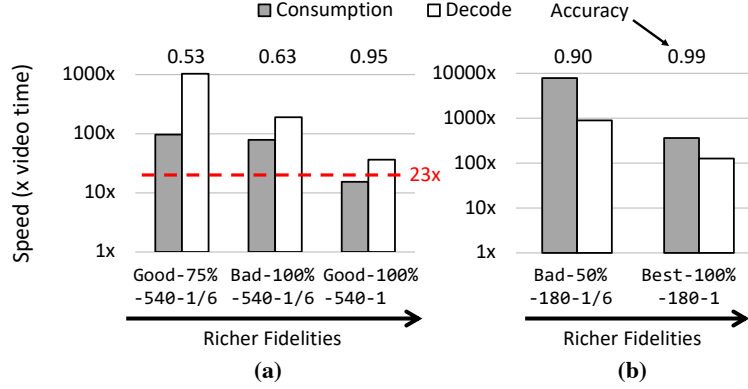


Figure 3.5. Video retrieval *could* bottleneck consumption. This is exemplified by the decoding speed vs. consumption speed comparisons for two different operators. **(a)** Operator: License. Consumption can be faster than decoding (speed shown as the dashed line), if the on-disk video is stored with the richest fidelity as ingested. Yet, consumption is still slower than decoding video of the same fidelity (white columns). **(b)** Operator: Motion. Consumption is faster than decoding, even if the on-disk video is of the same fidelity as consumed. Operator accuracy annotated on the top. See Section 3.5 for test hardware.

- **Consumption format:** the system supplies raw frame sequences to different operators running at a variety of accuracy levels, i.e., consumers. The format of each raw frame sequence is characterized by a fidelity option f . We refer to $CF\langle f \rangle$ as a consumption format.

We refer to the global set of video formats as the store’s *configuration* of video formats.

Configuration requirements These formats should jointly meet the following requirements:

R1. Satisfiable fidelity To supply frames in a consumption format $CF\langle f \rangle$, the system must retrieve video in storage format $SF\langle f, c \rangle$, where f is richer than or the same as f .

R2. Adequate retrieving speed Video retrieval should not slow down frame consumption. Figure 3.5 show two cases where the slowdown happens. a) For fast operators sparsely sampling video data, decoding may not be fast enough if the on-disk video is in the original format as it is ingested (e.g., 720p at 30 fps as from a surveillance camera). These consumers benefit from storage formats that are cheaper to decode, e.g., with reduced fidelity. b) For some operators quickly scanning frames looking for simple visual features,

even the storage format that is cheapest to decode (i.e., f' is the same as f ; cheapest coding option) is too slow. These consumers benefit from retrieving raw frames from disks.

R3. Consolidating storage formats Each stored video version incurs ingestion and storage costs. The system should exploit a key opportunity: creating one storage format for supplying data to multiple consumers, as long as satisfiable fidelity and adequate retrieving speed are ensured.

R4. Operating under resource budgets The store should keep the space cost by all videos under the available disk space. It should keep the ingestion cost for creating all video versions under the system’s transcoding bandwidth.

3.2.2 Inadequacy of existing video stores

Computer vision research typically assumes all the input data present in memory as raw frames, which does not hold for retrospective analytics over large videos: a server with 100 GB DRAM holds no more than two hours of raw frames even in low fidelity (e.g., 360p at 30 fps). Most video stores choose video formats in ad hoc manners *without optimizing for analytics* [74]. On one extreme, many save videos in one unified format (e.g., the richest fidelity expected by all operators). This minimizes storage and ingestion costs while incurring high retrieval cost. As a result, data retrieval may bottleneck operators. On the other extreme, one may incarnate all the storage formats with the fidelity exactly matching consumer expectations. This misses the opportunities for consolidating storage formats and will lead to excessive storage and ingestion costs. We will evaluate these two alternatives in Section 3.5.

Layered encoding cannot simplify the problem. Layered encoding promises space efficiency: it stores one video’s multiple fidelity options as complementary layers [75]. However, layered encoding has important caveats. i) Each additional layer has non-trivial storage overhead (sometimes 40%–100%) [76] which may result in storage space *waste* compared to consolidated storage formats. ii) Decoding is complex and slow, due to the combination of layers and random disk access in reading the layers. iii) Invented two decades ago, its adoption and coding performance are yet to be seen. Even if it is eventually adopted and proven desirable, it would make the configuration more complex.

Table 3.2. The library of operators in the current VStore.

Op	Description
Diff	Difference detector that detects frame differences [8]
S-NN	Specialized NN to detect a specific object [8]
NN	Generic Neural Networks, e.g., YOLO [50]
Motion	Motion detector using background subtraction [64]
License	License plate detector [64]
OCR	Optical character recognition [64]
Opflow	Optical flows for tracking object movements [77]
Color	Detector for contents of a specific color [7]
Contour	Detector for contour boundaries [78]

3.3 The VStore design

3.3.1 Overview

VStore runs on one or over a few commodity servers. It depends on existing query executors, e.g., OpenALPR, and a pre-defined library of operators. From the executor, VStore expects an interface for executing individual operators for profiling, and a manifest specifying a set of option accuracies for each operator. Table 3.2 listed 9 operators that are supported by the current VStore prototype. VStore tracks the whole set of $\langle operator, accuracy \rangle$ tuples as *consumers*.

Operation During operation, VStore periodically updates its video format configuration. For each ingested video stream, it periodically profiles operators and encoding/decoding, e.g., on a 10-second clip per hour. VStore splits and saves video footage in segments, which are 8-second video clips in our implementation. VStore retrieves or deletes each segment independently.

Challenges The major challenges are in configuration. i) Exhaustive search is infeasible. A configuration consists of a set of consumption formats from the 4D space \mathbb{F} and a set of storage formats from the 7D space $\mathbb{F} \times \mathbb{C}$. In our prototype, the total possible global configurations are 24^{15150} . Exhaustive profiling is expensive, as will be discussed in Section 3.3.2 iii) Optimizing for multiple resource types further complicates the problem.

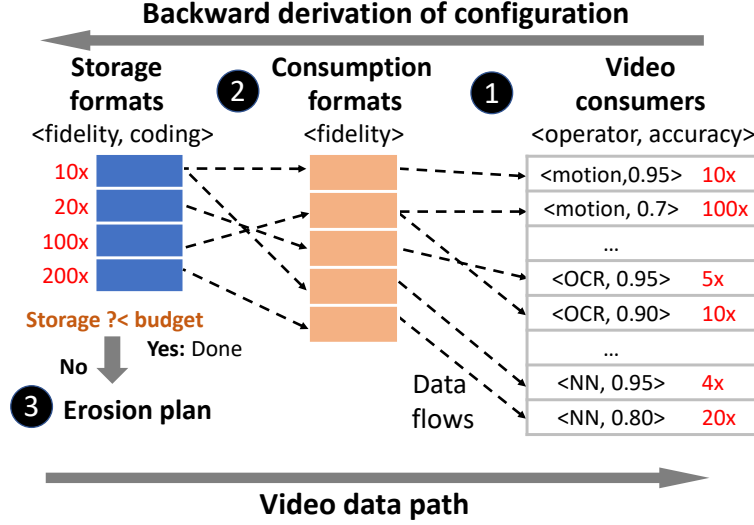


Figure 3.6. VStore derives the configuration of video formats. Example consumption/retrieval speed is shown.

These challenges were unaddressed. Some video query engines seek to ease configuration and profiling (challenge i and ii), but are limited to a few knobs [6], [9]. For the extensive set of knobs we consider, some of their assumptions, e.g., knob independence, do not hold. They optimize for one resource type – GPU cycles for queries, without accounting for other critical resources, e.g., storage (challenge 3).

Mechanism overview – backward derivation VStore derives the configuration *backwards*, in the direction opposite to the video data flow – from sinks, to retrieval, and to ingestion/storage. This is shown in Figure 3.6 ①–③. In this backward derivation, VStore optimizes for different resources in a progressive manner.

① Section 3.3.2: From all given consumers, VStore derives video consumption formats. Each consumer consumes, i.e., subscribes to, a specific consumption format. In this step, VStore optimizes data consumption speed.

② Section 3.3.3: From the consumption formats, VStore derives storage formats. Each consumption format subscribes to one storage format (along the reversed directions of dashed arrows in Figure 3.6). The chosen storage formats ensure i) satisfiable fidelity: a storage format SF has richer fidelity than any of its downstream consumption formats (CFs); ii) adequate retrieval speed: the retrieval speed of SF should exceed the speed of any downstream

consumer (following the dashed arrows in Figure 3.6). In this step, VStore optimizes for storage cost and keeps ingestion cost under budget.

③ Section 3.3.4: From all the derived storage formats, VStore derives a data erosion plan, gradually deleting aging video. In this step, VStore reduces storage cost to be under budget.

Limitations VStore treats individual consumers as independent without considering their dependencies in query cascades. If consumer A always precedes B in all possible cascades, the speed of A and B should be considered in conjunction. This requires VStore to model all possible cascades, which we consider as future work. VStore does not manage algorithmic knobs internal to operators [6], [9]; doing so would allow new, useful trade-offs for consumption but not for ingestion, storage, or retrieval.

3.3.2 Configuring consumption formats

Objective For each consumer $\langle op, accuracy \rangle$, the system decides a consumption format $\langle f_0 \rangle$ for the frames supplied to op . By consuming the frames, op should achieve the target accuracy while consuming data at the highest speed, i.e., with a minimum consumption cost. The primary overhead comes from operator profiling. Recall the relation $f \rightarrow \langle consumption\ cost, accuracy \rangle$ has to be profiled per operator regularly. For each profiling, the store prepares sample frames in fidelity f , runs an operator over them, and measures the accuracy and consumption speed. If the store profiles all the operators over all the fidelity options, the total number of required profiling runs, even for our small library of 9 operators is 2.7K. The total profiling time will be long, as we will show in the evaluation.

Key ideas VStore explores the fidelity space efficiently and only profiles a small subset of fidelity options. It works based on two key observations. **O1. Monotonic impacts** Increase in any fidelity knob leads to non-decreasing change in consumption cost and operator accuracy – richer fidelity will neither reduce cost nor accuracy. This is exemplified in Figure 3.3 showing the impact of changes to individual knobs. **O2. Image quality does not impact consumption cost.** Unlike other fidelity knobs controlling data quantity, image quality often does not affect operator workload and thus the consumption cost, as shown in Figure 3.3(b).

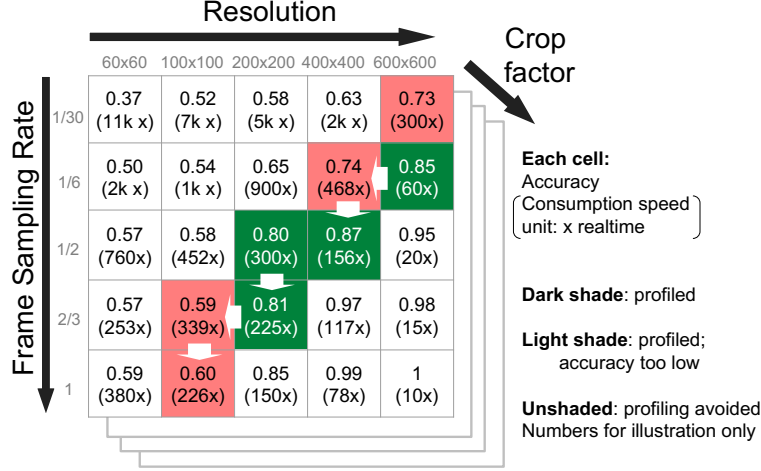


Figure 3.7. Search in a set of 2D spaces for a fidelity option with accuracy ≥ 0.8 and max consumption speed (i.e., min consumption cost).

We next sketch our algorithm deciding the consumption format for the consumer $\langle op, accuracy-t \rangle$: the algorithm aims finding f_0 that leads to accuracy higher than accuracy- t (i.e., adequate accuracy) with the lowest consumption cost.

Partitioning the 4D space i) Given that image quality does not impact consumption cost (O2), VStore starts by temporarily fixing the image quality knob at its highest value. ii) In the remaining 3D space (crop factor \times resolution \times sampling rate), VStore searches for fidelity f_0 that leads to adequate accuracy and the lowest consumption cost. iii) As shown in Figure 3.7, VStore partitions the 3D space into a set of 2D spaces for search. To minimize the number of 2D spaces under search, VStore partitions along the shortest dimension, chosen as the crop factor which often has few possible values (3 in our implementation). iv) The fidelity f_0 found from the 3D space already leads to adequate accuracy with the lowest consumption cost. While lowering the image quality of f_0 does not reduce the consumption cost, VStore still keeps doing so until the resultant accuracy becomes the *minimum* adequacy. It then selects the knob values as f_0 . This reduces other costs (e.g., storage) opportunistically.

Efficient exploration of a 2D space The kernel of the above algorithm is to search each 2D space (resolution \times sampling rate), as illustrated in Figure 3.7. In each 2D space, VStore looks for an *accuracy boundary*. As shown as shaded cells in the figure, the accuracy boundary splits the space into two regions: all points on the left have *inadequate* accuracies,

while all on the right have *adequate* accuracies. To identify the boundary, VStore leverages the fact that accuracy is monotonic along each dimension (O1). As shown in Figure 3.7, it starts from the top-right point and explores to the bottom and to the left. VStore only profiles the fidelity options on the boundary. It dismisses points on the left due to inadequate accuracies. It dismisses any point X on the right because X has fidelity richer than one boundary point Y; therefore, X incurs no less consumption cost than Y.

This exploration is inspired by a well known algorithm in searching in a monotone 2D array [79]. However, our problem is different: f_0 has to offer both adequate accuracy and lowest consumption cost. Therefore, VStore has to explore the entire accuracy boundary: its cannot stop at the point where the minimum accuracy is found, which may not result in the lowest consumption cost.

Cost & further optimization Each consumer requires profiling runs as many as $O((N_{sample} + N_{res}) * N_{crop} + N_{quality})$, where N_x is the number of values for knob x . This is much lower than exhaustive search which requires $(N_{sample}N_{res}N_{crop}N_{quality})$ runs. Furthermore, in profiling for the same operator’s different accuracies, VStore memoizes profiling results. Our evaluation will show that profiling *all* accuracies of one operator is still cheaper than exhaustively profiling the operator over the entire fidelity space.

What if a higher dimensional fidelity space? The above algorithm searches in the 4D space of the four fidelity knobs we consider. One may consider additional fidelity knobs (e.g., color channel). To search in such a space, we expect partitioning the space along shorter dimensions to still be helpful; furthermore, the exploration of 2D space can be generalized for higher dimensional spaces, by retrofitting selection in a high-dimensional monotonic array [79], [80].

3.3.3 Configuring storage formats

Objective For the chosen consumption formats and their downstream consumers, VStore determines the storage formats with satisfiable fidelity and adequate retrieval speed.

Enumeration is unaffordable One may consider enumerating all possible ways to partition the set of consumption formats (CFs), and determining a common storage format

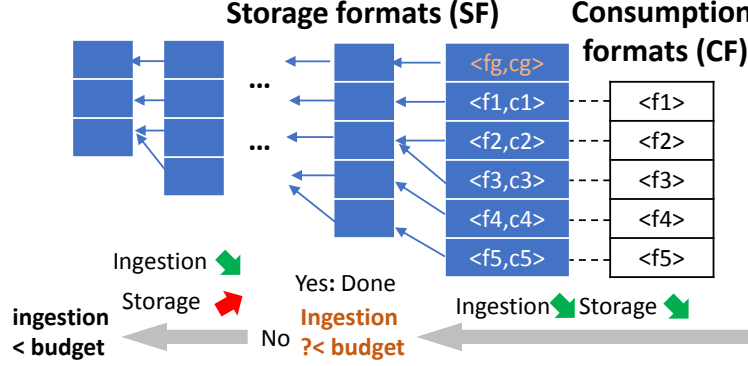


Figure 3.8. Iterative coalescing of storage formats.

for each subset of CFs. This enumeration is very expensive: the number of possible ways to partition a CF set is 4×10^6 for 12 CFs, and 4×10^{17} for the 24 CFs in our implementation [81], [82].

Algorithm sketch VStore coalesces the set of storage formats iteratively. Show on the right side of Figure 3.8, VStore starts from a full set of storage formats (SFs), each catering to a CF with identical fidelity. In addition, VStore creates a *golden* storage format SF_g $\langle fg, cg \rangle$: fg is the knob-wise maximum fidelity of all CFs; cg is the slowest coding option incurring the lowest storage cost. The golden SF is vital to data erosion to be discussed in Section 3.3.4. All these SFs participate in coalescing.

How to coalesce a pair? VStore runs multiple rounds of pairwise coalescing. To coalesce $SF_0 \langle f_0, c_0 \rangle$ and $SF_1 \langle f_1, c_1 \rangle$ into $SF_2 \langle f_2, c_2 \rangle$, VStore picks f_2 to be the knob-wise maximum of f_0 and f_1 for satisfiable fidelity. Such coalescing impacts resource costs in three ways. i) It reduces the ingestion cost as the video versions are fewer. ii) It may increase the retrieval cost, as SF_2 with richer fidelity tends to be slower to decode than SF_0/SF_1 . VStore therefore picks a cheaper coding option (c_2) for SF_2 , so that decoding SF_2 is fast enough for all previous consumers of SF_0/SF_1 . Even if the cheapest coding option is not fast enough, VStore bypasses coding and stores raw frames for SF_2 . iii) The cheaper coding in turn may increase storage cost.

How to select the coalescing pair? Recall that the goal of coalescing is to bring the ingestion cost under the budget. We explore two alternative approaches.

- ***Distance-based selection.*** As this is seemingly a hierarchical clustering problem, one may coalesce formats based on their similarity, for which a common metric is Euclidean distance. To do so, one may normalize the values of each knob and coalesce the pair of two formats that have the shortest distance among all the remaining pairs.
- ***Heuristic-based selection.*** We use the following heuristics: first harvesting “free” coalescing opportunities, and then coalescing at the expense of storage. Figure 3.8 illustrates this process. From the right to the left, VStore first picks up the pairs that can be coalesced to reduce ingestion cost *without* increasing storage cost. Once VStore finds out coalescing any remaining pair would *increase* storage cost, VStore checks if the current total ingestion cost is under budget. If not, VStore attempts to pick up cheaper coding options and continues to coalesce at the expense of increased storage cost, until the ingestion cost drops below the budget.

Overhead analysis The primary overhead comes from profiling. Being simple, distance-based selection incurs lower overhead: for each round, it only profiles the ingestion cost of the coalesced SF. Given that VStore coalesces at most N rounds (N being the number of CFs), the total profiling runs are $\min(\mathcal{O}(N), |\mathbb{F} \times \mathbb{C}|)$.

By comparison, heuristic-based selection tests all possible pairs among the remaining SFs in each round; for each pair, VStore profiles a video sample with the would-be coalesced SF, measuring decoding speed and the video sample size. The total profiling runs are $\min(\mathcal{O}(N^3), |\mathbb{F} \times \mathbb{C}|)$. In our implementation, N is 24 and $|\mathbb{F} \times \mathbb{C}|$ is 15K. Fortunately, by memoizing the previously profiled SFs in the same configuration process, VStore can significantly reduce the profiling runs, as we will evaluate in the evaluation. Furthermore, we will show that heuristic-based selection produces much more compact SFs.

3.3.4 Planning age-based data erosion

Objective In previous steps, VStore plans multiple storage formats of the same content catering to a wide range of consumers. In the last step, VStore reduces the total space cost to be below the system budget.

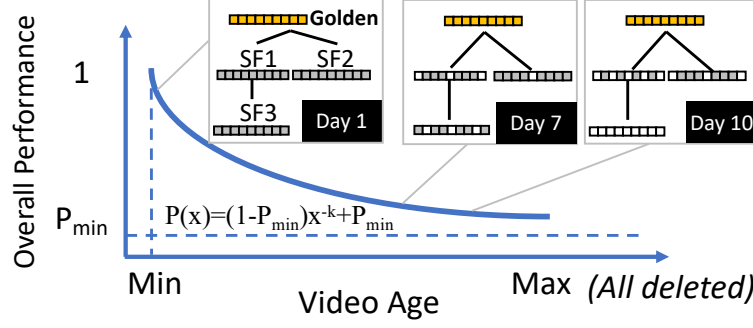


Figure 3.9. Data erosion decays operator speed and keeps storage cost under budget. Small cells: video segments.

Our insight is as follows. As video content ages, the system may slowly give up some of the formats, freeing space by relaxing the requirement for adequate retrieving speed on aged data (Section sec:case, R2). We made the following choices. i) **Gracefully degrading consumption speed.** VStore controls the rate of decay in speed instead of in storage space, as operator speed is directly perceived by users. ii) **Low aging cost.** VStore avoids transcoding aged videos which compete for encoder with ingestion. It hence creates no new storage formats for aging. iii) **Never breaks fidelity satisfiability.** VStore identifies some video versions as fallback data sources for others, ensuring all consumers to achieve their desired accuracies as long as the videos are still in lifespan.

Data erosion plan VStore plans erosion at the granularity of video ages. Recall that VStore saves video as segments on disks (each segment contains 8-second video in our implementation). As shown in Figure 3.9, for each age (e.g., per day) and for each storage format, the plan dictates the percentage of deleted segments, which accumulate over ages.

How to identify fallback video formats? VStore organizes all the storage formats of one configuration in a tree, where the edges capture *richer-than* relations between the storage formats, as shown in Figure 3.9. Consumers, in an attempt to access any deleted segments of a child format, fall back to the parent format (or even higher-level ancestors). Since the parent format offers richer fidelity, the consumers are guaranteed to meet their accuracies; yet, the parent’s retrieval speed may be inadequate to the consumers (e.g., due to costlier decoding), thus decaying the consumers’ *effective* speed. If a consumer has to consume a fraction p of segments from the parent format, on which the effective speed is

only a fraction α of its original speed with no eroded data, the consumer’s relative speed is defined as the ratio between its decayed speed to its original, given by $\alpha/((1 - p)\alpha + p)$. VStore never erodes the golden format at the root node; with its fidelity richer than any other format, the golden format serves as the ultimate fallback for all consumers.

How to quantify the overall speed decay? Eroding one storage format may decay the speeds of multiple consumers to various degrees, necessitating a global metric for capturing the overall consumer speed. Our rationale is for all consumers to fairly experience the speed decay. Following the principle of max-min fairness [83], we therefore define the overall speed as the minimum relative speed of all the consumers. By this definition, the overall speed P is also relative, in the range of $(0,1]$. P is 1 when the video content is the youngest and all the versions are intact; it reaches the minimum P_{min} when all but the golden format are deleted.

How to set overall speed target for each age? We follow the power law function, which gives gentle decay rate and has been used on time-series data [65]. In the function $P(x) = (1 - P_{min})x^{-k} + P_{min}$, x is the video age. When $x = 1$ (youngest video), P is 1 (the maximum overall speed); as x grows, P approaches P_{min} . Given a decay factor k (we will show how to find a value below), VStore uses the function to set the target overall speed for each age in the video lifespan.

How to plan data erosion for each age? For gentle speed decay, VStore always deletes from the storage format that would result in the minimum overall speed reduction. In the spirit of max-min fairness, VStore essentially spreads the speed decay evenly among consumers.

VStore therefore plans erosion by resembling a fair scheduler [84]. For each video age, i) VStore identifies the consumer Q that currently has the lowest relative speed; ii) VStore examines all SFs in the “richer-than” tree, finding the one that has the least impact on the speed of Q ; iii) VStore plans to delete a fraction of segments from the found format, so that another consumer R ’s relative speeds drops below Q ’s. VStore repeats this process until the overall speed drops below the target of this age.

Summary VStore generates an erosion plan by testing different values for the decay factor k . It finds the lowest k (most gentle decay) that brings down the total storage cost

accumulated over all video ages under budget. For each tested k , VStore generates a tentative plan: it sets speed targets for each video age based on the power law, plans data erosion for each age, sums up the storage cost across ages, and checks if the storage cost falls below the budget. As higher k always leads to lower total storage cost, VStore uses binary search to quickly find a suitable k .

3.4 Implementation

We built VStore in C++ and Python with 10K SLoC. Running its configuration engine, VStore orchestrates several major components.

Coding and storage backend: VStore invokes FFmpeg, a popular software suite for coding tasks. VStore’s ingestion uses the libx264 software encoder; it creates one FFmpeg instance to transcode each ingested stream. Its retrieval invokes NVIDIA’s NVDEC decoder for efficiency. VStore invokes LMDB, a key-value store [85], as its storage backend. VStore stores 8-second video segments in LMDB. We choose LMDB as it well supports MB-size values.

Ported query engines: We ported two query engines to VStore. We modify both engines so they retrieve data from VStore and provide interfaces for VStore’s profiling. OpenALPR [64] recognizes vehicle license plates. Its operators build on OpenCV and run on CPU. To scale up, we create a scheduler that manages multiple OpenALPR contexts and dispatches video segments. NoScope [8] is a recent research engine. It combines operators that execute at various speeds and invoke deep NN. It invokes TensorFlow [86] as the NN framework, which runs on GPU.

Operator lib: The two query engines provide 6 operators as shown in Figure 3.1. In particular, S-NN uses a very shallow AlexNet [87] produced by NoScope’s model search and NN uses YOLOv2 [50].

3.5 Evaluation

We answer the following questions in evaluation:

§3.5.2: Does VStore provide good end-to-end results?

§3.5.3: Does VStore adapt configurations to resource budgets?

§3.5.4: Does VStore incur low overhead in configuration?

3.5.1 Methodology

Video Datasets We carried out our evaluation on six videos, extensively used as benchmarks in prior work [6]–[8], [15]. We include videos from both dash cameras (which contain high motion) and surveillance cameras that capture traffic from heavy to light. The videos are: *jackson*, from a surveillance camera at Jackson Town Square; *miami*, from a surveillance camera at Miami Beach crosswalk; *tucson*: from a surveillance camera at Tucson 4-th Avenue. *dashcam*, from a dash camera when driving in a parking lot; *park*, from a stationary surveillance camera in a parking lot; *airport*, from a surveillance camera at JAC parking lot. The ingestion formats of all videos are 720p at 30 fps encoded in H.264.

VStore setup We, as the system administrator, declare a set of accuracy levels $\{0.95, 0.9, 0.8, 0.7\}$ for each operator. These accuracies are typical in prior work [6]. In determining F1 scores for accuracy, we treat as the ground truth when the operator consumes videos in the ingestion format, i.e., highest fidelity. In our evaluation, we run the two queries as illustrated in Figure 3.1: Query A (Diff + S-NN + NN) and query B (Motion + License + OCR). In running the queries, we, as the users, select specific accuracy levels for the operators of the query. In running queries, we, as the users, specify different accuracy levels for the constituting operators. We run query A on the first three videos and B on the remainder, as how these queries are benchmarked in prior work [8], [64]. To derive consumption formats, VStore profiles the two sets of operators on *jackson* and *dashcam*, respectively. Each profiled sample is a 10-second clip, a typical length used in prior work [6]. VStore derives a unified set of storage formats for all operators and videos.

Hardware environment We test VStore on a 56-core Xeon E7-4830v4 machine with 260 GB DRAM, 4×1TB 10K RPM SAS 12Gbps HDDs in RAID 5, and a NVIDIA Quadro P6000 GPU. By their implementation, the operators from ALPR run on the CPU; we limit them to use up to 40 cores for ensuring the query speed comparable to commodity multi-core servers. The operators from NoScope run on the GPU.

Table 3.3. A sample configuration of video formats automatically derived by VStore.

(a): All consumption formats for all operators (columns) at different accuracy levels (rows). Total 21 unique.

Each cell shows: **fidelity**, subscribed storage format **SF** and **consumption speed**.

(b): All storage formats. Each cell shows: **fidelity**, coding (kFrameInt-SpeedStep), coalesced video size (per sec), and **retrieval speed**.

1. RAW frames are in YUV420p pixel format.

2. RAW frames can be sampled individually from disk, thus the range of retrieval speed.

Note: Above tables show an example of derived CFs and SFs. Operators in Query A (Diff + S-NN + NN) are profiled on jackson, and operators in Query B (Motion + License + OCR) are profiled on dashcam. CFs and SFs might differ across different videos.

	Diff	S-NN	NN	Motion	License	OCR	Storage Formats (SFs)
F1=0.95	best-100p-2/3-75% SF3 3211x	best-200p-1-50% SF3 600x	good-600p-2/3-100% SFg 4x	bad-144p-1/30-75% SF3 25134x	best-540p-1-100% SFg 10x	best-720p-1/2-100% SFg 11x	SFg best-720p-1-100% 250-slowest 1393KB 23x
F1=0.90	best-60p-2/3-75% SF3 4587x	best-200p-1/2-75% SF3 1630x	good-600p-2/3-75% SFg 5x	bad-180p-1/30-50% SF3 26117x	best-540p-1/2-100% SFg 20x	best-540p-1/2-100% SFg 13x	SF1 good-540p-1/6-100% 250-slowest 409KB 178x
F1=0.80	best-200p-1/30-100% SF3 30585x	best-200p-1/2-50% SF3 3680x	good-400p-1/30-100% SF2 120x	bad-180p-1/30-50% SF3 26117x	good-540p-1/6-100% SF1 62x	best-540p-1/30-100% SF2 165x	SF2 best-540p-1/30-100% 10-fast 92KB 331x
F1=0.70	best-60p-1/30-75% SF3 34132x	best-200p-1/6-75% SF3 8102x	good-400p-1/30-75% SF2 134x	bad-180p-1/30-50% SF3 26117x	good-540p-1/30-75% SF2 314x	good-540p-1/30-100% SF2 165x	SF3 best-200p-1-100% RAW 1843KB ¹ 1137-34132x ²

(a) Consumption Formats (CF)

(b) Storage Formats (SF)

3.5.2 End-to-end results

Configuration by VStore VStore automatically configuring video formats based on its profiling. Table 3.3 shows a snapshot of configuration, including the whole set of consumption formats (CFs) and storage formats (SFs). For all the 24 consumers (6 operators at 4 accuracy levels), VStore generates 21 unique CFs, as shown in Table 3.3(a). The configuration has 109 knobs over all 21 CFs (84 knobs) and 4 SFs (25 knobs), with each knob having up to 10 possible values. Manually finding the optimal combination would be infeasible, which warrants VStore’s automatic configuration. In each column (a specific operator), although the knob values *tend* to decrease as accuracy drops, the trend is complex and can be non-monotone. For instance, in column Diff, from F1=0.9 to 0.8, VStore advises to decrease sampling rate, while *increase* the resolution and crop factor. This reflects the complex impacts of knobs as stated in Section 3.1.4. We also note that VStore chooses extremely low fidelity for Motion at all accuracies ≤ 0.9 . It suggests that Motion can benefit from an even larger fidelity space with even cheaper fidelity options.

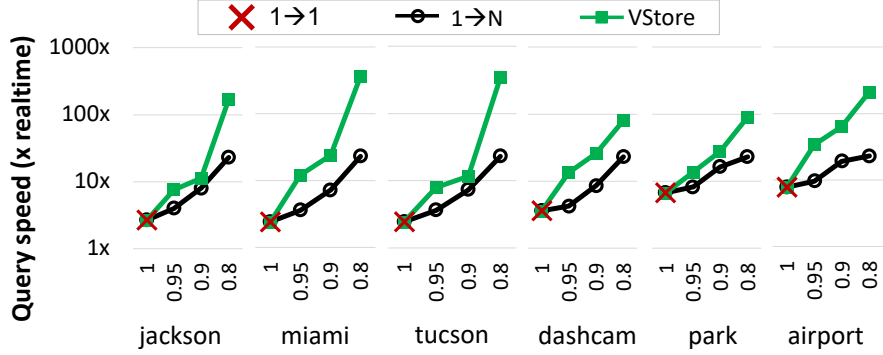
From the CFs, VStore derives 4 SFs, including one golden format (SF_g), as listed in Table 3.3(b). Note that we, as the system admin, has not yet imposed any budget on ingestion cost. Therefore, VStore by design chooses the set of SFs that minimize the total storage cost (Section 3.3.3). The CF table on the left tags each CF with the SF that each CF subscribes to. As shown, the CFs and SFs jointly meet the design requirements

R1–R3 in Section 3.3.3: each SF has fidelity richer than/equal to what its downstream CFs demand; the SF’s retrieval speed is always faster than the downstream’s consumption speed. Looking closer at the SFs: SF_g mostly caters to consumers demanding high accuracies but low consumption speeds; SF3, stored as low-fidelity raw frames, caters to high-speed consumers demanding low image resolutions; between SF_g and SF3, SF1 and SF2 fill in the wide gaps of fidelity and costs. Again, it is difficult to manually determine such a complementary set of SFs without VStore’s configuration.

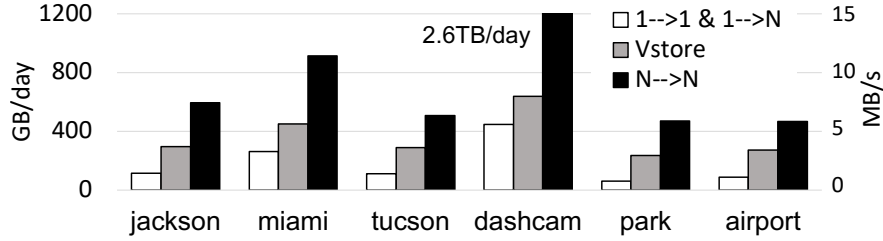
Alternative configurations We next quantitatively contrast VStore with the following alternative configurations:

- $1 \rightarrow 1$ stores videos in the golden format (SF_g in Table 3.3). All consumers consume videos in this golden format. This resembles a video database oblivious to algorithmic consumers.
- $1 \rightarrow N$ stores videos in the golden format SF_g . All consumers consume video in the CFs determined by VStore. This is equivalent to VStore configuring video formats for consumption but not for storage. The system, therefore, has to decode the golden format and convert it to various CFs.
- $N \rightarrow N$ stores videos in 21 SFs, one for each unique CF. This is equivalent to VStore giving up its coalescing of SFs.

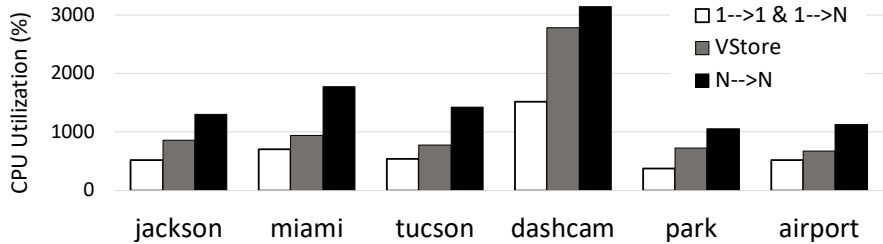
Query speed As shown in Figure 3.10(a), VStore achieves good query speed overall, up to $362\times$ realtime. VStore’s speed is incomparable with performance reported for retrospective analytics engines [7], [8]: while VStore streams video data (raw/encoded) from disks through decoder to operators, the latter were tested with all input data preloaded as raw frames in memory. VStore offers flexible accuracy/cost trade-offs: for queries with lower target accuracies, VStore accelerates query speed by up to $150\times$. This is because VStore elastically scales down the costs: according to the lower accuracies, it switches the operators to CFs that incur lower consumption cost; the CFs subscribe to SFs that incur lower retrieval cost.



(a) Query speeds (y-axis; logarithmic scale) as functions of target operator accuracy (x-axis). Query A on left 3 videos; query B on right 3 videos. By avoiding video retrieval bottleneck, VStore significantly outperforms others. In this work, we assume the ingestion format is the ground truth (accuracy = 1). Note that N→N is omitted since the speed is the same as VStore.



(b) Storage cost per video stream, measured as the growth rate of newly stored video size as ingestion goes on. VStore's coalescing of SFs substantially reduces storage cost.



(c) Ingestion cost per video stream, as required CPU usage for transcoding the stream into storage formats. VStore's SF coalescing substantially reduces ingestion cost. Note that this shows VStore's *worst-case* ingestion cost with no ingestion budget specified; see Table 3.4 for more.

Figure 3.10. End-to-end result of VStore.

Figure 3.10(a) also shows the query speed under alternative configurations. 1→1 achieves the best accuracy (treated as the ground truth) as it consumes video in the full fidelity as ingested. However, it cannot exploit accuracy/cost trade-offs, offering a fixed operating point. By contrast, VStore offers extensive trade-offs and speeds up queries by two orders of magnitude.


1→N customizes consumption formats for consumers while only storing the golden format. Although it minimizes the consumption costs for consumers, it essentially caps the effective speed of all consumers at the speed of decoding the golden format, which is about 23× of realtime. The bottlenecks are more serious for lower accuracy levels (e.g., 0.8) where many consumers are capable of consuming data as fast as tens of thousand times of realtime, as shown in Table 3.3(a). As a result, VStore outperforms 1→N by 3×-16×, demonstrating the necessity of the SF set.

Storage cost Figure 3.10(b) compares the storage costs. Among all, N→N incurs the highest costs, because it stores 21 video versions in total. For *dashcam*, a video stream with intensive motion which makes video coding less effective, the storage cost reaches as high as 2.6 TB/day, filling a 10TB hard drive in four days. In comparison, VStore consolidates the storage formats effectively and therefore reduces the storage cost by 2×-5×. 1→1 and 1→N require the lowest storage space as they only save one video version per ingestion stream; yet, they suffer from high retrieval cost and low query speed.

Ingestion cost Figure 3.10(c) demonstrates that VStore substantially reduces ingestion cost through consolidation of storage formats. Note that it shows VStore’s *worst-case* ingestion cost. As stated earlier, in the end-to-end experiment with no ingestion budget imposed, VStore, therefore, reduces the ingestion cost without *any* increase in the storage cost. As we will show next, once an ingestion budget is given, VStore can keep the ingestion cost much lower than the worst case with only a minor increase in storage cost.

Overall, on most videos VStore requires around 9 cores to ingest one video stream, transcoding it into the 4 SFs in real time (30 fps). Ingesting *dashcam* is much more expensive, as the video contains intensive motion. VStore’s cost is 30%-50% lower than N→N, which must transcode each stream to 21 SFs. 1→1 and 1→N incur the lowest ingestion cost as

Table 3.4. In response to ingestion budget drop, VStore tunes coding and coalesces formats to stay under the budget with increase in storage cost. Changed knobs shown in red.

		Budget Reduces 				
Cores for ingest		>=7	6	3	2	1
Stor.	MB/sec	3.039	3.042	3.094	3.273	3.561
	GB/day	250.4	250.7	254.9	269.7	293.4
Storage Fmts	SFg	250-slowest	250-slowest	250- <u>slow</u>	250- <u>med</u>	250- <u>fast</u>
	SF1	250-slowest	250- <u>slow</u>	250-slow	250- <u>med</u>	250- <u>fast</u>
	SF2	10-fast	10-fast	10-fast	10-fast	250-fast
	SF3	RAW	RAW	RAW	RAW	RAW

Coding option: “Keyframe Interval” - “SpeedStep”

they only transcode the ingestion stream to the golden format, yet at the expense of costly retrieval and slow query speed.

3.5.3 Adapting to resource budgets

Ingestion budget VStore elastically adapts its configuration with respect to the ingestion budget. To impose budget, we, as the system admin, cap the number of CPU cores available to one FFmpeg that transcodes each ingested stream. In response to the reduced budget, VStore gently trades off storage for ingestion. Table 3.4 shows that as the ingestion budget drops, VStore incrementally tunes up the coding speed (i.e., cheaper coding) for individual SFs. As a trade-off, the storage cost slowly increases by 17%. During this process, the increasingly cheaper coding *overprovisions* the retrieval speed to consumers and therefore will never fail the latter’s requirements. Note that at this point, the total ingestion output throughput is still less than 3.6 MB/s; even the system ingests 56 streams with its 56 cores concurrently, the required disk throughput 200 MB/s is still far below that of a commodity HDD array (1 GB/s in our platform).

We also find out that SFs as well as the ingestion cost quickly plateaus as VStore’s library includes more operators. Figure 3.11 shows how the ingestion cost increases as operators are sequentially added, following the order listed in Table 3.2, to VStore’s library. The ingestion

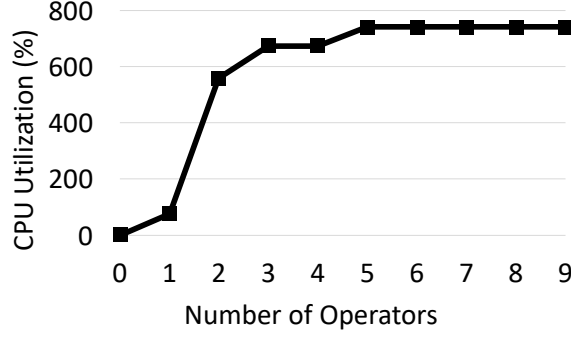


Figure 3.11. Transcoding cost does not scale up with the number of operators. Operator sequence follows Table 3.2.

cost stabilizes as the number of operators exceeds 5, as additional operators share existing SFs.

Storage budget VStore’s data erosion effectively respects the storage budget with gentle speed decay. To test VStore’s erosion planning, we, as system admin, set the video lifespan to 10 days; we then specify different storage budgets.

With all 4 SFs listed in Table 3.3(b), 10-day video stream will take up 5 TB of disk space. If we specify a budget above 5 TB, VStore will determine not to decay ($k=0$), shown as the flat line in Figure 3.12(a). Further reducing the storage budget prompts data erosion. With a 4 TB budget, VStore decays the overall operator speed (defined in Section 3.3.4) following a power law function ($k=1$). As we further reduce the budget, VStore plans more aggressive decays to respect the budget. Figure 3.12(b) shows how VStore erodes individual storage formats under a specific budget. On day 1 (youngest), all 4 SFs are intact. As the video ages, VStore first deletes segments from SF1 and SF2 that have lower impacts on the overall speed. For segments older than 5 days, VStore deletes all the data in SF1-3, while keeping the golden format intact (not shown).

3.5.4 Configuration overhead

VStore incurs moderate configuration overhead, thanks to our techniques in Section 3.3.2 and Section 3.3.3. Overall, one complete configuration (including all required profiling) takes

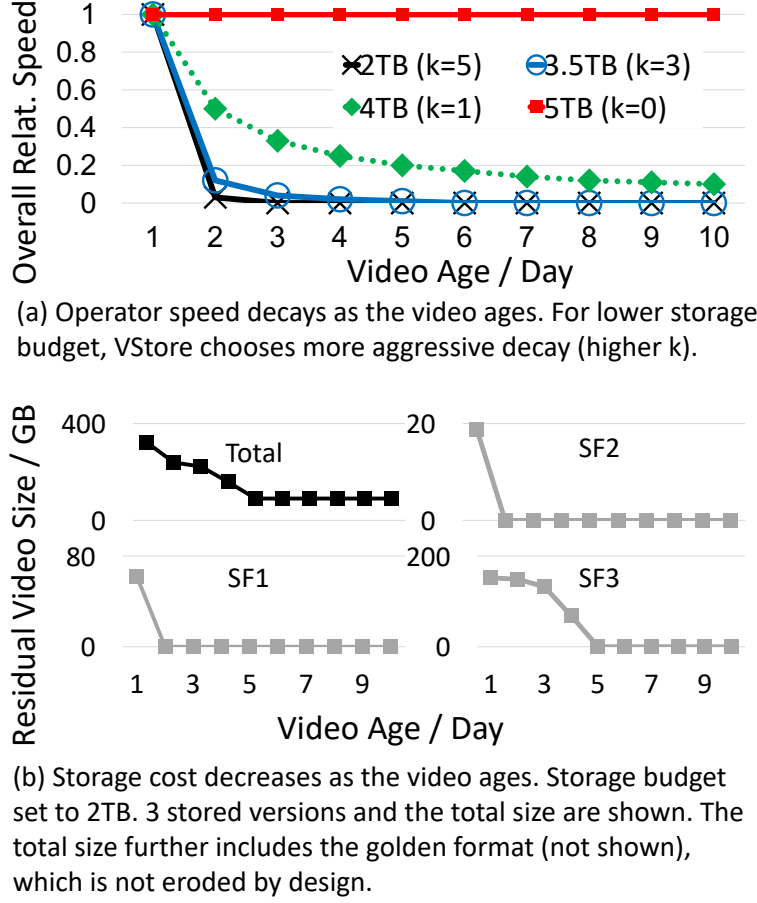


Figure 3.12. Age-based decay in operator speed (a) and reducing storage cost (b) to respect storage budget.

around 500 seconds, suggesting the store can afford one configuration process in about every 1 hour online.

Configuring consumption formats Figure 3.13 shows the overhead in determining consumption formats. Compared to exhaustive profiling of all fidelity options, VStore reduces the number of profiling runs by $9\times$ – $15\times$ and the total profiling delay by $5\times$, from 2000 seconds to 400 seconds. We notice that the License operator is slow, contributing more than 75% of total delay, likely due to its CPU-based implementation.

Configuring storage formats We have validated that VStore is able to find resource-efficient storage formats as exhaustive enumeration does.

Heuristic-based selection: We first test heuristic-based selection for producing SFs (Section 3.3.3). We compare it to exhaustive enumeration, on deriving SFs from the 12 CFs

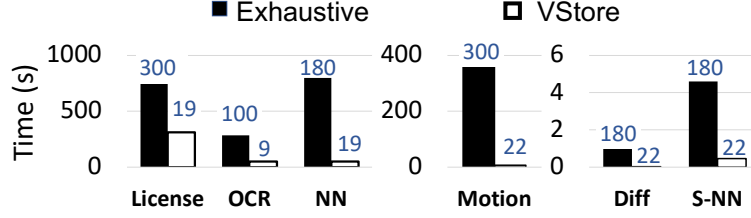


Figure 3.13. Time spent on deriving consumption formats. Numbers of profiling runs are annotated above columns. Each required profiling runs on a 10-second video segment. VStore reduces overhead by $5\times$ in total.

used in query B; we cannot afford more CFs would which make exhaustive enumeration very slow. Both methods result in identical storage formats, validating VStore’s rationale behind coalescing. Yet, VStore’s overhead (37 seconds) is 2 orders of magnitude lower than enumeration (5548 seconds).

To derive the storage formats from all the 21 unique consumption formats in our evaluation, VStore incurs moderate absolute overhead (less than 1 min) too. Throughout the 17 rounds of coalescing, it only profiles 475 (3%) storage formats out of all 15K possible ones. We observed that its memorization is effective: despite 5525 storage formats are examined as possible coalescing outcomes, 92% of them have been memoized before and thus requires no new profiling.

Distance-base selection: We then test the other strategy. We use Euclidean distance as the similarity metric. The configuration takes only 18 seconds, $2\times$ shorter than the heuristic-based selection mentioned above. This is because calculating the distances requires no expensive profiling as heuristic-based selection does.

Comparison of resultant SFs: The two strategies also derive very different SF sets: while the SFs derived by heuristic-based selection is close to optimal as shown above, the SFs derived by distance-based selection incur $2.2\times$ higher storage cost. This is because the latter strategy, while simple, overlooks the fact that different knobs have complex and varying resource impacts (Section 3.1.4), which cannot be simply normalized across knobs.

3.6 Discussion

Adapting to changes in operators and hardware VStore works with any possible queries composed by operators/accuracies pre-defined in its library (Section 3.1.2). If users add a new operator (or a new accuracy level), VStore would need to profile the new operator and derive corresponding CFs for it. If users change the platform hardware (e.g., adding a new GPU), VStore would need to re-profile all existing operators. Conceptually, this also triggers an update to the SFs. Since transcoding existing on-disk videos is expensive, VStore only applies the updated SFs to forthcoming videos; for existing videos, VStore makes each new CF subscribe to the cheapest existing SF with satisfiable fidelity (Section 3.2.1). As a result, on existing videos, operators run with designated accuracies, albeit slower than optimal. As this portion of videos age and retire, operators run at optimal speed on all videos.

Qualitative comparison against Focus [15] As stated in Section 3.1.2, Focus by design is limited to fixed query pipelines – object detection consisting of one cheap neural network (NN) and one full NN. This contrasts with VStore which supports diverse queries. Nevertheless, we compare their resource costs on such an object detection pipeline.

Ingestion cost. VStore continuously runs transcoding. As already shown in Figure 3.11, the transcoding cost quickly plateaus as the number of operators grows. While the current VStore prototype runs transcoding on CPU for development ease, low-cost hardware transcoder is pervasive: recent work showcases a transcoder farm of \$20 Raspberry Pis, with each device transcoding 4 video streams in real time (720×480 at 30 fps) [88], [89]. We, therefore, estimate the hardware cost for each video ingestion to be less than a few dozen dollars.

By comparison, at ingestion time, Focus continuously runs the cheap NN on GPU. On a high-end GPU (Tesla P100, ~\$4,000), the cheap NN is reported to run at 1.92K fps; assuming perfect scalability, this GPU supports up to 60 video streams. The hardware investment for ingesting each video stream is around \$60, which is 2×-3× higher than VStore. If the ingested streams are fewer (e.g., several or a few dozen as typical for a small deployment), the GPU is underutilized, which further increases per-stream investment. Running the ingestion on

public cloud helps little: Amazon EC2’s single-GPU instance (P3) costs nearly \$17.5K per year [90].

Query cost. At query time, VStore would run the cheap NN on all frames and the full NN on the frames selected by the cheap NN. By comparison, Focus only runs the full NN on the frames selected by the cheap NN (it already runs the cheap NN at ingestion). The comparison between VStore’s query cost and that of Focus depends on two factors: (i) the frame selectivity f and (ii) the ratio α between the full NN speed and the cheap NN speed. Therefore, the ratio between VStore’s query delay and that of Focus is given by $r = 1 + \alpha/f$. With the NNs used by Focus, $\alpha = 1/48$ [15].

When the frame selectivity is low, e.g., the queried objects are sparse in the video, VStore’s query delay is significantly longer (e.g., when $f = 1\%$, $r = 3$). However, as the selectivity increases, the query delay difference between VStore and Focus quickly diminishes, e.g., when $f = 10\%$, $r = 1.2$; when $f = 50\%$, $r = 1.04$. Furthermore, as the speed gap between the two NNs enlarges, e.g., with an even cheaper NN, the query delay difference quickly diminishes as well.

3.7 Related work

Optimizing video analytics Catering to retrospective video analytics, BlazeIt [7] proposes a query model and corresponding execution techniques [7]. NoScope [8] reduces query cost with cheap early filters before expensive NN. To run NNs on mobile devices, MCDNN [91] trades off between accuracy and resource constraints by model compression.

Optimizing live video analytics For distributed, live video analytics, VideoStorm [9] and VideoEdge [92] search for best knobs and query placements over clusters to meet accuracy/delay requirements. For live video analytics on the edge, LAVEA[35] and Vigil [13] partitions analytics pipelines between the edge and the cloud. Jain *et al.* [57] optimize video analytics over multiple cameras through cross-camera correlations. Pakha *et al.* [93] co-tune network protocols with video analytics objectives, e.g., accuracy. However, all the systems are incapable of optimizing ingestion, storage, retrieval, and consumption in conjunction.

Video/image storage Facebook’s Haystack [94] accelerates photo access through meta-data lookups in main memory. Intel’s VDMS [95], [96] accelerates image data access through a combination of graph-based metadata and array-based images backed by TileDB [97]. They focus on images rather than videos. Targeting NN training, NVIDIA’s Video Loader [98] (a wrapper over NVDEC and FFmpeg) optimizes random loads of encoded video frames. To support video analytics at scale, Scanner [99] organizes video collections and raster data as tables in a data store and executes costly pixel-level computations in parallel. All these systems are short on controlling visual data formats according to analytics. NVIDIA DeepStream SDK [100] supports video frames flow from GPU’s built-in decoders to stream processors without leaving the GPU. It reduces memory move, but no fundamental change in trade-offs between retrieval and consumption.

Time-series database Recent time-series data stores co-design storage format with queries [65], [101]. However, the data format/schema (timestamped sensor readings), the operators (e.g., aggregation), and the analytics structure (no cascade) are different from video analytics. While some databases [102], [103] provide benefits on data aging or frequent queries, they could not make storage decisions based on video queries as they are oblivious to the analytics.

Multi-query optimization Relational databases [104] and streaming databases [105], [106] enable sharing data and computation across queries with techniques such as scan sharing [107]–[109]. By doing so, they reduce data move in memory hierarchy and coalesce computation across queries. VStore, in a similar fashion, support data sharing among multiple possible queries, albeit at configuration time instead of at run time. By doing so, VStore coalesces data demands across queries/operators and hence reduces the ingestion and storage cost. Through load shedding [110]–[112], streaming databases trade accuracy for lower resource consumption; VStore makes similar trade-offs for vision operators.

Video systems for human consumers Many multimedia server systems in 90’s stored videos on disk arrays in multiple resolutions or in complementary layers, in order serve human clients [113], [114]. Since then, Kang *et al.* [21] optimizes placement of on-disk video layers in order to reduce disk seek. Oh *et al.* [115] segments videos into shots, which are easier for humans to browse and search. Recently, SVE [116] is a distributed service for

fast transcoding of uploaded videos in data centers. ExCamera [73] uses Amazon lambda function for parallel video transcoding. These systems were not designed for, and therefore are oblivious to, algorithmic consumers. They cannot automatically control video formats for video analytics.

3.8 Conclusions

VStore automatically configures video format knobs for retrospective video analytics. It addresses the challenges by the huge combinatorial space of knobs, the complex knobs impacts, and high profiling cost. VStore explores a key idea called backward derivation of configuration: the video store passes the video quantity and quality desired by analytics backward to retrieval, to storage, and to ingestion. VStore automatically derives complex configurations. It runs queries as fast as up to $362\times$ of video realtime.

4. DIVA: SUPPORTING EXPLORATORY VIDEO QUERIES ON ZERO-STREAMING CAMERAS

4.1 Background & motivations

4.1.1 Cold videos are already pervasive

Case study: Cold videos in real-world deployment We conduct an IRB-approved study examining existing camera deployment on a campus. Spanning 1 mi², the campus hosts tens of thousands of employees and operates more than 1,000 cameras. All captured videos are stored for a few months for retrospective queries before deletion. The camera deployment supports AI-based queries, e.g., object detection, *not* traceable to unique persons, and reviews by human analysts. We analyzed system logs spanning six continuous months: in over 3,000,000 hours of videos (5.4 PB) have been captured, only **<0.005%** video data from **<2%** cameras are queried.

Why are most videos cold? (1) Interesting video events are both unpredictable (thus the need for capturing excessive videos) and sparse (thus low chances for footage being queried). For example, severe traffic breakdown contributes to less than 5% of the time per day [117]; Foreign Intelligence Surveillance Court only reviewed a tiny fraction of video for terrorism events [118]. (2) Analyzing videos is expensive: it still requires a GPU of a few thousand dollars for high-accuracy object detection over a video stream [8]. (3) In years to come, cheap cameras will produce more videos.

4.1.2 Target queries and their execution

We target ad-hoc queries [8], [13], [15], [18]. The query parameters, including object classes, video timespans, and expected accuracies, are specified at query time rather than video capture time. Such queries are known for flexibility.

High-accuracy object detection is essential Object detection is the core of ad-hoc queries [7]. Minor accuracy loss in object detection may result in substantial loss in query performance, as we will demonstrate in Section 4.7. While NNs significantly advance object detection, new models with higher accuracy demand much more compute. For instance,

compared to YOLOv3 (2018) [10], CornerNet (2019) [119] improves Average Precision by 28% while being $5\times$ more expensive.

Low-cost cameras cannot answer queries without cloud Cameras in real-world deployment are reported to be resource-constrained [11]. Low-cost cameras ($< \$40$) have wimpy cores, e.g., Cortex-A9 cores for YI Home Camera [23] and MIPS32 cores for Wyze-Cam [22]; their DRAM is no more than a few GBs [120], [121]. In recent benchmarks, they run state-of-the-art object detection at 0.1 FPS [122], [123], incapable of keeping up with video capture at 1–30 FPS [8], [15]. NN accelerators still cannot run high-accuracy object detection fast enough at low enough monetary cost, e.g., Intel’s Movidius (\$70) runs YOLOv3 at no faster than 0.5 FPS. We expect that the resource gap between high-accuracy object detection and low-cost camera continues to exist in the future.

4.1.3 A case for zero streaming

Streaming cold videos wastes bandwidth As discussed in Section 1.2, cameras are cheap while wireless spectrum is precious. Deploying streaming cameras on a shared network incurs poor experience [124], [125] and draws researcher attention [12], [13]. Dedicated networks are costly [27] and thus only suit a small number of cameras in critical locations. While wireless bandwidth grows, consumer demand grows even faster, e.g., $20\times$ for VR/AR and $10\times$ for gaming [126]. Cold video traffic should not contend with consumers for network bandwidth.

Streaming optimizations cannot offset the waste One may reduce FPS or resolution of streamed videos. Even if users tolerate the resultant lower query accuracy, the saved bandwidth is incomparable to the waste on overwhelmingly streamed cold videos, as we will experimentally show (Section 4.7). On-camera “early filters” [11], [12], [14] are still suboptimal when querying massive *cold* videos. (1) Without knowing query objects/parameters at video capture time, a camera may run a generic filter, e.g., discarding no-motion frames; it still streams substantial survival frames (e.g., consider a street-view camera). As stated above, most of these frames will remain cold and hence wasted. (2) The camera may run a

Table 4.1. Cheap μ SD cards on cameras retain long videos for humans to review [128] or for machines to analyze [15].

Size	Yr.2017	Yr.2020	720p@30FPS	720p@1FPS
128GB	\$45	\$17	~ 11 days	~ 3 weeks
256GB	\$150	\$28	~ 3 weeks	~ 6 weeks

large set of specific filters covering all possible query objects/parameters. Even if possible, this incurs a much higher compute cost to camera.

Edge processing does not justify streaming Cameras may stream to edge servers. Yet, streaming hundreds if not thousands of *always-on*, *cold* video streams, even if possible on certain wireless infrastructures, still wastes precious wireless spectrum at the edge [127]. Furthermore, deploying and managing video edge servers can be challenging and costly in many scenarios, such as construction sites and remote farms.

Camera can retain videos long enough Table 4.1 shows the price of μ SD cards has been dropped by $2.6\times$ – $5.4\times$ in the past few years. Cameras can retain videos for several weeks and for several months soon. Such retention periods are adequate for most retrospective query scenarios, in which videos are stored from a few weeks to a few months based on legal regulations [28]–[31] and best practice. For privacy, many regulations *prohibit* video retention longer than a few months and mandate deletion afterwards [30], [31].

Our model & design scope To harness cold videos, we advocate for zero streaming. We focus on cold videos being queried for the first time and querying individual cameras. We intend our design to form the basis of future enhancement, e.g., caching for repetitive queries, exploiting past queries for refinement [129], and exploiting cross-camera topology [56]. We address limited compute resource on cameras [121] and limited network bandwidth [26]. We do not consider the cloud as a limiting factor, assuming it runs fast enough to process frames uploaded from cameras.

4.2 Overview

Query types Concerning a specific camera, an ad-hoc query $(\mathcal{T}, \mathcal{C})$ covers a video timespan \mathcal{T} , typically hours or days, and an object class \mathcal{C} as detectable by modern NNs, e.g., any

Table 4.2. A summary of supported queries. \mathcal{T} is the queried video timespan; \mathcal{C} is the queried object class.

Type & Semantics	Execution	User’s view of query results	Performance Metrics
Retrieval. Get positive video frames (i.e., containing \mathcal{C}) within \mathcal{T}	Camera: multipass ranking of frames Uploaded: ranked frames Cloud: object detection for identifying true positives	<ul style="list-style-type: none"> • Positive frames being uploaded; • Estimated % of positives retrieved 	The rate of the user receiving positive frames
Tagging. Get time ranges from \mathcal{T} that contain \mathcal{C}	Camera: multipass filtering of frames Uploaded: unresolved frames; tags of resolved frames Cloud: object detection to tag unresolved frames	<ul style="list-style-type: none"> • A video timeline with pos/neg ranges; • Tagging resolution, i.e., 1 in every N adjacent frames tagged 	The refining rate of tagging resolution seen by the user
Counting. Get max/mean/median count of \mathcal{C} across all frames in \mathcal{T}	Camera: multipass ranking (max) or random sampling (mean/median) of frames Uploaded: ranked or sampled frames Cloud: object detection to count objects	<ul style="list-style-type: none"> • Running counts that converge to ground truth; • % of frames processed; • Estimated time to complete the query 	The rate of running counts converging to ground truth

of the 80 classes of YOLOv3 [10]. As summarized in Table 4.2, DIVA supports three query types: **Retrieval**, e.g., “retrieve all images that contain buses from yesterday”; **Tagging**, e.g., “return all time ranges when any deer shows up in the past week”, in which the time ranges are returned as metadata but not images; **Counting**, e.g., “return the maximum number of cars that ever appear in any frame today”.

System components DIVA spans a camera and the cloud. Between them, the network connection is only provisioned at query time. To execute a query, a camera runs lightweight NNs, or operators, to *filter* or *rank* the queried frames for upload. On the uploaded frames, the cloud runs generic, high-accuracy object detection and materializes query results. Table 4.2 summarizes executions for different queries:

- The camera executes **rankers** for *Retrieval* and *max Count* queries. A ranker scores frames; a higher score suggests that a frame is more likely to contain *any* object of interest (for Retrieval) or *a large count* of such objects (for max Count).
- The camera executes **filters** for *Tagging* queries. A filter scores frames; it resolves any frame scored below/above two pre-defined thresholds as negative/positive, and deems other frames as unresolved. For each resolved frame, the camera uploads a positive/negative tag; the camera either uploads unresolved frames for the cloud to decide or defer them to more accurate filters on camera in subsequent passes.

Query execution Upon receiving a query, the cloud retrieves all landmarks in queried video as low-resolution thumbnails, e.g., 100×100, with object labels and bounding boxes (Figure 4.1 ①). The cloud uses landmarks: (1) *to estimate object spatial distribution*,

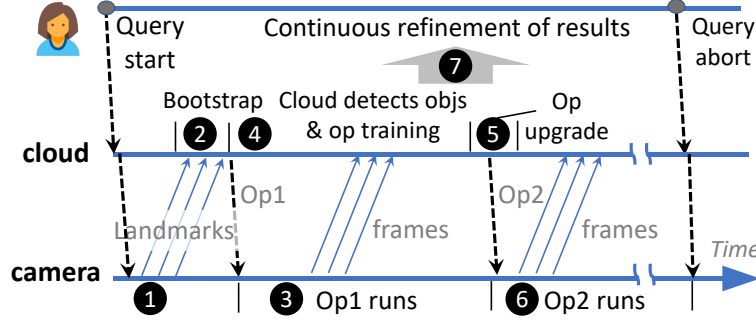


Figure 4.1. The workflow of DIVA’s query execution.

e.g., “90% queried objects appear in a 100×100 region on the top-right”, which is crucial to query optimization (Section 4.3); (2) *as the initial training samples* for bootstrapping a family of camera operators (2). The camera filters/ranks frames and uploads the ranked or surviving frames (3). The cloud processes the uploaded frames and emits results, e.g., positive frames. It trains operators for higher accuracy (4). Observing resource conditions and positive ratios in uploaded frames, the cloud upgrades the operator on camera (5). With the upgraded operator, the camera continues to process remaining frames (6). Step (4)–(6) repeat until query abort or completion. Throughout the query, the cloud keeps refining the results presented to the user (7).

Notable designs (1) The camera processes frames in multiple passes, one operator in each pass. (2) The camera processes and uploads frames asynchronously. For instance, when the camera finishes ranking 100 out of total 1,000 frames, it may have uploaded the top 50 of the 100 ranked frames. This is opposed to common ranking which holds off frame upload until all the frames are ranked [130]–[132]. (3) The processing/upload asynchrony facilitates video exploration: it amortizes query delay over many installments of results; it pipelines query execution with user thinking [33]. Table 4.2 summarizes a user’s view of query results and the performance metrics. While such online query processing has been known [133], [134], we are the first applying it to visual data.

Limitations DIVA is not designed for several cases and may underperform: querying very short video ranges, e.g., minutes, for which simply uploading all queried frames may

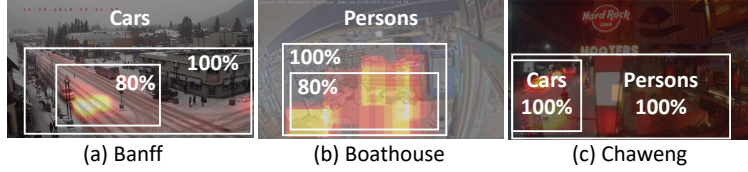


Figure 4.2. Class spatial skews in videos. In (a) Banff: 80% and 100% of cars appear in regions that are only 19% and 57% of the whole frame, respectively.



Figure 4.3. Class spatial distribution can be estimated from sparse frames sampled over long video footage. Among the three heatmaps: while sparse sampling over short footage (left) significantly differs from dense sampling of long footage (right), sparse sampling of long footage (middle) is almost equivalent to Video: Tucson (see Table 4.3).

suffice without operators; querying non-stationary cameras for which landmarks may not yield accurate object distribution.

4.3 Landmark design

Surveillance cameras have a unique opportunity: to learn *object class distribution* from weeks of videos. We focus on **spatial skews**: objects of a given class are likely to concentrate on certain small regions on video frames. In examples of Figure 4.2(a)-(b), most cars appear near a stop sign; most persons appear in a shop’s aisle. To our knowledge, such long-term skews are untapped in prior computer vision work, which focused on minute-long videos [5], [56], [92], [135], [136].

We have three key observations. (1) One object class may exhibit different skews in different videos (Figure 4.2(a)-(c)); different classes may exhibit different skews in the same video (Figure 4.2(c)). (2) The skews are pervasive: surveillance cameras cover long time spans and a wide field of view, where objects are small; in the view, objects are subject to social constraints, e.g., buses stop at traffic lights, or physical constraints, e.g., humans appear on

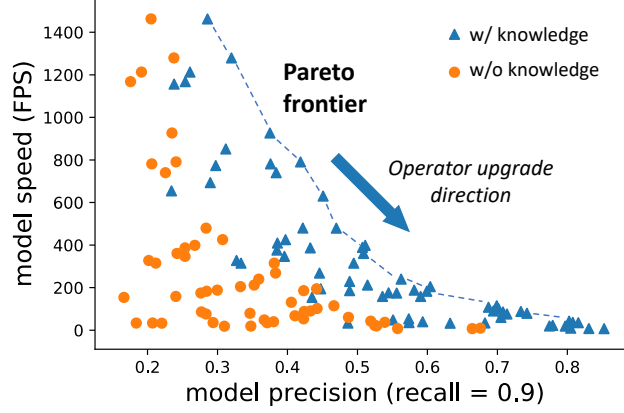


Figure 4.4. On-camera operators benefit from long-term video knowledge substantially. Each marker: an operator. For querying buses on video Banff (see Table 4.3).

the floor. (3) The skews can be learned through *sparse* frame samples, as exemplified by Figure 4.3.

To exploit such an opportunity, DIVA makes the following design choices. **(1) High-accuracy object detection:** at capture time, the camera runs an object detector with the highest accuracy as allowed by the camera’s hardware, mostly memory capacity. This is because camera operators crucially depend on the correctness of landmarks, i.e., the object labels and bounding boxes. We will validate this experimentally (Section 4.7.2). **(2) Sparse sampling at regular intervals:** to accommodate slow object detection on cameras, the camera creates landmarks at long intervals, e.g., 1 in every 30 seconds in our prototype (Section 4.7). Sparse sampling is proven valid for estimating statistics of low-frequency signals [137], e.g., object occurrence in videos in our case. We will validate this (Section 4.7.2); without assuming a priori of object distribution, regular sampling ensures unbiased estimation of the distribution [138]. Given a priori, cameras may sample at corresponding random intervals for unbiased estimation.

Key idea: exploiting spatial skews for performance The cloud learns the object class distribution from landmarks of the queried video timespan. It generates a heatmap for spatial distribution (Figure 4.2). Based on the heatmap, the cloud produces camera operators consuming frame regions of different locations and sizes. Take Figure 4.2(a) as an example: a

filter may consume bottom halves of all frames and accordingly filter frames with no cars; for Figure 4.2(b), a ranker may consume a smaller bounding box where 80% persons appear and rank frames based on their likelihood of containing more persons. Figure 4.4 shows that, by zooming into smaller regions, operators run faster and deliver higher accuracy. By varying input region locations/sizes, DIVA produces a set of operators with diverse costs/accuracies. By controlling the execution order of operators, DIVA processes “popular” frame regions prior to “unpopular” regions. DIVA never omits any region when it executes a query to completion to guarantee correctness.

What happens to instances uncaptured by landmarks? Sparse by design, landmarks are not meant to capture all object instances; instead, they are used as inexact estimators and initial training samples. Reducing landmarks will degrade query speed, as we will experimentally quantify in Section 4.7.2. Doing so, however, does not affect query correctness or accuracy: the instances uncaptured by landmarks will be eventually processed by DIVA as a query goes on.

4.4 Online operator upgrade

4.4.1 The rationale

Three factors determine a query’s execution speed:

1. *Pending workloads*: the difficulty of the frames to be processed, i.e., how likely will the frame be mis-filtered or mis-ranked on camera.
2. *Camera operators*: cheap operators spend less time on each frame but are more likely to mis-filter/mis-rank frames, especially difficult frames. This is shown in Figure 4.4.
3. *Network condition*: the available uplink bandwidth.

The three factors interplay as follows.

- **Queries executed with on-camera *rankers*** A camera ranks and uploads frames asynchronously (Section 4.2). The key is to maximize the rate of true positive frames arriving at the cloud, for which the system must balance ranking speed/accuracy with

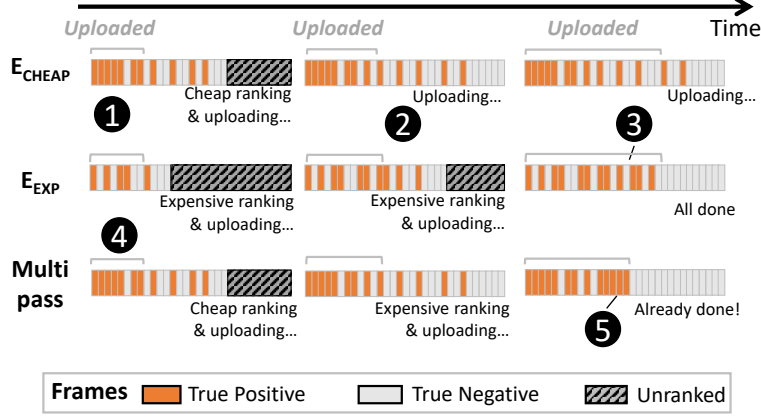


Figure 4.5. Three alternative executions of a Retrieval query, showing multipass ranking (bottom) outperforms running individual rankers alone (top two). Each row: snapshots of the upload queue at three different moments. In a queue: ranking/uploading frames from left to right.

upload bandwidth. (1) When the camera runs a *cheaper* ranker, it ranks frames at a much higher rate than uploading the frames; as a result, the cloud receives frames *hastily* selected from a *wide* video timespan. (2) When the camera runs an *expensive* ranker, the cloud receives frames selected *deliberately* from a *narrow* timespan. (3) The camera should never run rankers slower than upload, which is as bad as uploading unranked frames. As an example, E_{CHEAP} and E_{EXP} on the top of Figure 4.5 compare two possible executions of the same query, running cheap/expensive rankers respectively. Shortly after the query starts (①), E_{CHEAP} swiftly explores more frames on camera; it outperforms E_{EXP} by discovering and returning more true positive frames. As both executions proceed to harder frames (②), E_{CHEAP} makes more mistakes in ranking; it uploads an increasingly large ratio of negatives which wastes the execution time. By contrast, E_{EXP} ranks frames slower yet with much fewer mistakes, hence uploading fewer negatives. It eventually returns all positives earlier than E_{CHEAP} (③).

The microbenchmark in Figure 4.6(a) offers quantitative evidence. E_1 spends *less* time ($0.7\times$) in returning the first 90% positives, but *more* time ($1.7\times$) in returning 99% positives. Furthermore, *lower* upload bandwidth favors a more *expensive* ranker,

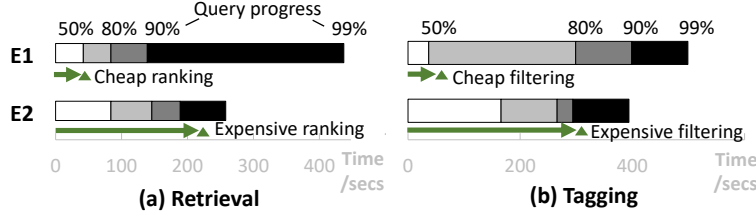


Figure 4.6. Cheap/expensive camera operators excel at different query stages. Each subfigure: two alternative executions of the same query, showing query progress (bars) and the corresponding operator’s progress (arrows).

as the uploaded frames would contain a *higher* ratio of positives, better utilizing the precious bandwidth.

- **Queries executed with on-camera *filters*** The key is to maximize the rate of resolving frames on camera. Cheap filters excel on easy frames, resolving these frames fast with confidence. They are incapable on difficult frames, wasting time on attempting frames without much success in resolving. They would underperform *expensive* filters that spend more time per frame yet being able to resolve more frames.

The benchmark in Figure 4.6(b) shows two executions with cheap/expensive filters. Early in the query, E1 makes faster progress as the camera quickly resolves 50% of the frames (4× less time than E2). Later in the execution, E1 lags behind as the camera cannot resolve the remaining frames and must upload them. By contrast, E2 resolves 82% of frames on camera and only uploads the remaining 18%. As a result, E2 takes 1.3× less time in completing 90% and 99% of the query.

Summary & implications It is crucial for DIVA to pick operators with optimal cost/accuracy at query time. The choice not only varies across queries but also varies throughout a query’s execution: easy frames are processed early, leaving increasingly difficult frames that call for more expensive operators. DIVA should monitor pending frame difficulty and network bandwidth and upgrade operators accordingly.

4.4.2 Multipass, multi-operator execution

DIVA manages operators with the following techniques.

- A camera processes frames iteratively with multiple operators.
- The cloud progressively updates operators on camera, from cheaper ones to more expensive ones, as the direction shown in Figure 4.4. In picking operators, the cloud dynamically adapts operator speed to frame upload speed.
- The cloud uses frames received in early execution stages to train operators for later stages; as the latter operators are more expensive, they require more training samples.

Multipass ranking This is exemplified by the bottom execution in Figure 4.5. The camera first runs a cheap ranker, moving positives towards the front of the upload queue (④). Subsequently, the camera runs an expensive ranker, continuously reordering unsent frames in a smaller scope (⑤). Throughout the query, the camera first quickly uploads easy frames that are quickly ranked and slows down to vet difficult frames with expensive/accurate ranking. Notably, the cheaper ranker roughly prioritizes the frames as input for the expensive ranker, ensuring the efficacy of the expensive ranker. In actual query executions, a camera switches among 4–8 operators (Section 4.7).

Multipass filtering The camera sifts undecided, unsent frames in multiple passes, each with a more expensive filter over a sample of the remaining frames. Throughout one query, early, cheaper filters quickly filter easier frames, leaving more difficult frames for subsequent filters to resolve.

4.5 Query execution planning

DIVA plans a concrete query execution by (1) the camera’s policy for selecting frames to process; (2) the cloud’s policy for upgrading on-camera operators. We now discuss them.

4.5.1 Executing *Retrieval* queries

Policy for selecting frames To execute the initial operator, the camera prioritizes fixed-length video spans (e.g., 1 hour) likely rich in positive frames, estimated based on landmark frames. In executing subsequent operators, the camera processes frames in their existing ranking as decided by earlier operators, as described in Section 4.4. The camera

gives opportunities to frames never ranked by prior operators, interleaving their processing with ranked frames with mediocre scores (0.5).

Policy for operator upgrade As discussed in Section 4.2, DIVA switches from cheap operators to expensive ones, and matches operator speed to frame upload rate. To capture an operator op 's *relative* speed to upload, it uses one simple metric: the ratio between the two speeds, i.e., $f_{op} = FPS_{op}/FPS_{net}$. Operators with higher f_{op} tend to rapidly *explore* frames while others tend to *exploit* slowly. The operator speed FPS_{op} is profiled offline. **(1)**

Selecting the initial operator In general, DIVA should fully utilize the upload bandwidth with positive frames. As positive frames are scattered in the queried video initially, the camera should explore all frames sufficiently fast. Otherwise, it would either starve the uplink or knowingly upload negative frames. Based on this idea, the cloud picks the most accurate operator from the ones that are fast enough, i.e., $f_{op} \times R_{pos} > 1$, where R_{pos} is the ratio of positives in the queried video, estimated from landmarks. **(2) When to upgrade:**

current operator losing its vigor The cloud upgrades operators either when the current operator finishes processing all frames, or the cloud observes a continuous quality decline in recently uploaded frames, an indicator of the current operator's incapability. To decide the latter, DIVA employs a rule: the positive ratio in recently uploaded frames are $k \times$ (default: 5) lower than the frames uploaded at the beginning; **(3) Selecting the next operator:**

slow down exponentially Since the initial operator promotes many positives towards the front of the upload queue, subsequent operators, scanning from the queue front, likely operate on a larger fraction of positives. Accordingly, the cloud picks the most accurate operator among much slower ones, s.t. $f_{op(i+1)} > \alpha \times f_{op(i)}$, where α controls speed decay in subsequent operators. A larger α leads to more aggressive upgrade: losing more speed for higher accuracy. In the current prototype, we empirically choose $\alpha = 0.5$. Since f is relative to FPS_{net} measured at every upgrade, the upgrade adapts to network bandwidth change *during* a query.

4.5.2 Executing *Tagging* queries

Recall that for *Tagging*, a camera runs multipass filtering; the objective of each pass is to tag, as positive (P) or negative (N), at least one frame from every K adjacent frames. We call K the group size; DIVA pre-defines a sequence of group sizes as refinement levels, e.g., $K = 30, 10, \dots, 1$. As in prior work [7], [8], [15], the user specifies tolerable error as part of her query, e.g., 1% false negative and 1% false positive; DIVA trains filters with thresholds to meet the accuracy.

Policy for selecting frames The goal is to quickly tag easy frames in individual groups while balancing the workloads of on-camera processing and frame upload. An operator op works in two stages of each pass. i) *Rapid attempting*. op scans all the groups; it attempts one frame per group; if it succeeds, it moves to the next group; it adds undecidable frames (U) to the upload queue. ii) *Work stealing*. op steals work from the end of upload queue. For an undecidable frame f belonging to a group g , op attempts other untagged frames in g ; once it succeeds, it removes f from the upload queue as f no longer needs tagging in the current pass. After one pass, the camera switches to the next refinement level (e.g., $10 \rightarrow 5$). It keeps all the tagging results (P, N, U) while cancels all pending uploads. It re-runs the frame scheduling algorithm until it meets the finest refinement level or query terminated.

Policy for operator upgrade Given an operator op and γ_{op} , the ratio of frames it can successfully tag, DIVA measures op 's efficiency by its effective tagging rate, $FPS_{op} \times \gamma_{op} + FPS_{net}$, as a sum of op 's successful tagging rate and the uploading rate. As part of operator training, the cloud estimates γ_{op} for all the candidate operators by testing them on all landmarks (early in query) and uploaded frames (later in query). To select every operator, initial or subsequent, the cloud picks the candidate with the highest effective tagging rate. The cloud upgrades operators either when the current operator has attempted all remaining frames or another candidate having an effective tagging rate $\beta \times$ or higher (default value 2).

4.5.3 Executing *Counting* queries

Max Count: Policy for selecting frames To execute the initial operator, the camera randomly selects frames to process, avoiding the worst cases that the max resides at the

end of the query range. For subsequent operators, the camera processes frames in existing ranking decided by earlier operators.

Max Count: Policy for operator upgrade As the camera runs rankers, the policy is similar to that for *Retrieval* with a subtle yet essential difference. To determine whether the current operator shall be replaced, the cloud must assess the quality of recently uploaded frames. While for *Retrieval*, DIVA conveniently measures the quality as the ratio of *positive* frames, the metric does not apply to *max Count*, which seeks to discover *higher* scored frames. Hence, DIVA adopts the Manhattan distance as a quality metric among the permutations from the ranking of the uploaded frames (as produced by the on-camera operator) and the ranking that is re-computed by the cloud object detector. A higher metric indicates worse quality hence more urgency for the upgrade.

Average/Median Count: no on-camera operators After the initial upload of landmarks, the camera randomly samples frames in queried videos and uploads them for the cloud to refine the average/median statistics. To avoid any sampling bias, the camera does not prioritize frames; it instead relies on the Law of Large Numbers (LLN) to approach the average/median ground truth through continuous sampling.

4.6 Implementation and methodology

Operators We architect on-camera operators as variants of AlexNet [87]. We vary the number of convolutional layers (2–5), convolution kernel sizes (8/16/32), the last dense layer’s size (16/32/64); and the input image size (25×25/50×50/100×100). We empirically select 40 operators to be trained by DIVA online; we have attempted more but see diminishing returns. These operators require small training samples (e.g., 10K images) and run fast on camera.

Background subtraction filters static frames at low overhead [15]. DIVA employs a standard technique [139]: during video capture, a camera detects frames that have little motion ($< 1\%$ foreground mask) and omits them in query execution. On our camera hardware (Table 4.4), background subtraction is affordable in real time during capture. For fair comparisons, we augment all baselines with background subtraction.

Table 4.3. 15 videos used for test. Each video: 720P at 1FPS lasting 48 hours. Column 1: video type. T – traffic; O/I – outdoor/indoor surveillance; W – wildlife.

	Name	Object	Description
T	JacksonH [140]	car	A busy intersection in Jackson Hole, WY
	JacksonT [141]	car	A night street in Jackson Hole, WY
	Banff [142]	bus	A cross-road in Banff, Alberta, Canada
	Mierlo [143]	truck	A rail crossing in Netherlands
	Miami [144]	car	A cross-road in Miami Beach, FL
	Ashland [145]	train	A level crossing in Ashland, VA
	Shibuya [146]	bus	An intersection in Shibuya, Japan
O	Chaweng [147]	bicycle	Absolut Ice Bar (outside) in Thailand
	Lausanne [148]	car	A pedestrian plaza in Lausanne, Switzerland
	Venice [149]	person	A waterfront walkway in Venice, Italy
	Oxford [150]	bus	A street beside Oxford Martin school, UK
	Whitebay [151]	person	A beach in Virgin Islands
I	CoralReef [152]	person	An aquarium video from CA
	BoatHouse [153]	person	A retail store from Jackson Hole, WY
W	Eagle [154]	eagle	A tree with an eagle nest in FL

Videos & Queries We test on 15 videos captured from 15 live camera feeds (Table 4.3). Each video lasts continuous 48 hours. We preprocess all videos to be 720P at 1 FPS, consistent with prior work [15]. We test *Retrieval/Tagging/Counting* queries on 6/6/3 videos. We intentionally choose videos with disparate characteristics and hence different degrees of difficulty. For each video, we pick a representative object class to query; across videos, these classes are diverse. For *Tagging*, we set query error to be $< 1\%$ FN/FP as prior work did [8]. Noting that the query “accuracy” is already implied in the query process, e.g., the fraction of positive frames retrieved.

Test platform & parameters As summarized in Table 4.4(a), we test on embedded hardware similar to low-cost cameras [120], [121]. We use Rpi3 as the default camera hardware and report its measurement unless stated otherwise. During query execution, both devices set up a network connection with 1MB/s default bandwidth to emulate typical WiFi condition [26]. Note that this network bandwidth is *not* meant for streaming; it is only for a camera while the camera is being queried. We run YOLOv3 as the high-accuracy object

Table 4.4. Experiment configurations.

(a) Hardware platforms.

Cameras	Rpi3 (default): Raspberry Pi 3 (\$35). 4xCortex-A53, 1GB DRAM Odroid : XU4 (\$49) 4xCortexA15 & 4xCortexA7, 2GB DRAM
CloudServer	2x Intel Xeon E5-2640v4, 128GB DRAM GPU: Nvidia Titan V

(b) DIVA and the baselines. The table summarizes their executions for capture and query. NNs: Yv3 – YOLOv3, high accuracy (mAP=57.9); YTiny – YOLOv3-tiny, low accuracy (mAP=33.1).

	Cam:Landmarks	Cam:Query	Cloud:Query
ClondOnly	–	Only upload frames	Yv3 on all uploaded frames
OptOp	Yv3 every 30 secs	Run one optimal op	
PreIndexAll	YTiny every sec	Parse YTiny result	
DIVA	Yv3 every 30 secs	Multi passes & ops	

detector on camera and cloud (Table 4.4(b)). We will study alternative models, landmarks, and resources in Section 4.7.2.

Baselines As summarized in Table 4.4(b), we compare DIVA with three alternative designs augmented with background subtraction and only process/transmit non-static video frames.

- **CloudOnly**: a naive design that uploads all queried frames at query time for the cloud to process.
- **OptOp**: in the spirit of NoScope [8], the camera runs only one ranker/filter specialized for a given query, selected by a cost model for minimizing full-query delay. To make OptOp competitive, we augment it with landmark Compared to DIVA, OptOp’s key differences are the lack of operator upgrade and the lack of operator optimization by long-term video knowledge.
- **PreIndexAll**: in the spirit of Focus [15], the camera runs a cheap yet generic object detector on all frames. We pick YOLOv3-tiny (much cheaper than YOLOv3) as the detector affordable by Rpi3 in real time (1 FPS). The detector plays the same role as an operator in DIVA, except that it runs at capture time: for *Retrieval* and *Counting*,

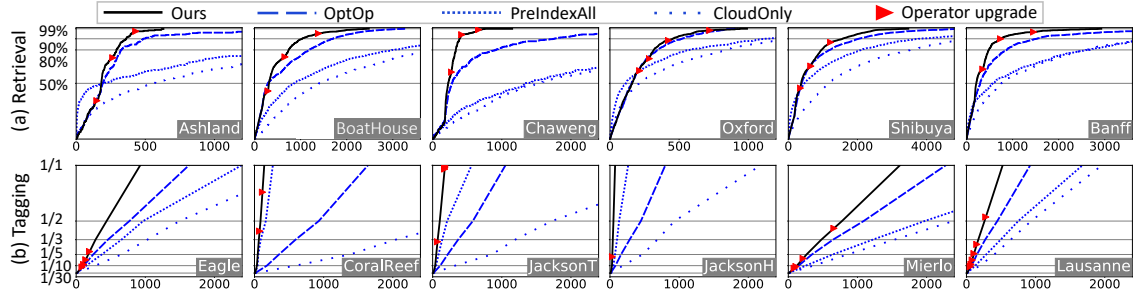


Figure 4.7. On *Retrieval* and *Tagging* queries, DIVA shows good performance and outperforms the alternatives. x-axis for all: query delay (secs). y-axis for (a): % of retrieved instances; for (b): refinement level ($1/N$ frames).

the detector’s output scores are used to prioritize frames to upload at query time; for *Tagging*, the output is used to filter the frames that have enough confidence. Compared to DIVA, *PreIndexAll*’s key differences are: it answers queries solely based on the indexes built at capture time; it requires no operator training or processing actual images at query time.

4.7 Evaluation

4.7.1 End-to-end performance

Full query delay is measured as: *Retrieval* – the time to receive 99% positive frames as in prior work [15]; *Tagging* – the time taken to tag every frame; *Counting* – the time to reach the ground truth (max) or converge within 1% error of the ground truth (avg/median). Overall, DIVA delivers good performance and outperforms the baselines significantly.

- *Retrieval* (Figure 4.7(a)). On videos each lasting 48 hours, DIVA spends $\sim 1,900$ seconds on average, i.e., $89\times$ of video realtime. On average, DIVA’s delay is $3.8\times$, $3.1\times$, and $2.0\times$ shorter than *CloudOnly*, *PreIndexAll*, and *OptOp*, respectively.
- *Tagging* (Figure 4.7(b)). DIVA spends ~ 581 seconds on average ($297\times$ realtime). This delay is $16.0\times$, $2.1\times$, and $4.3\times$ shorter than *CloudOnly*, *PreIndexAll*, and *OptOp*, respectively.
- *Counting* (Figure 4.8). DIVA’s average/median take several seconds to converge. For *average Count*, DIVA’s delay is $65.1\times$ and up to three orders of magnitude shorter

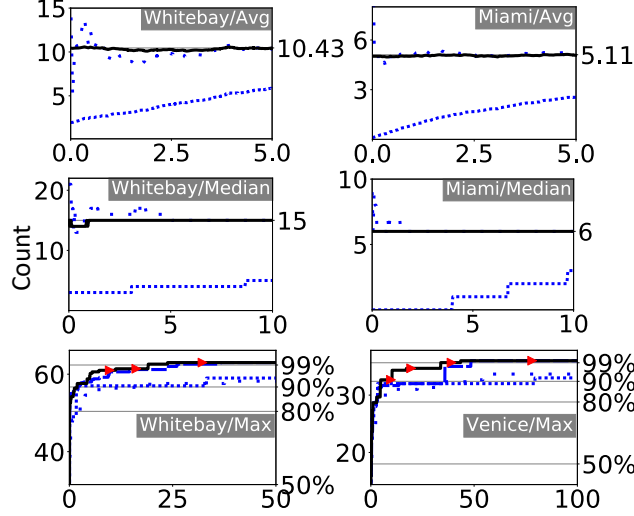


Figure 4.8. On *Counting* queries, DIVA shows good performance and outperforms the alternatives. Legend: see Figure 4.7. x-axis for all: query delay (secs). y-axis for left plots: count; for top two right plots: ground truth for avg/median queries; for bottom right plot: % of max value.

than **CloudOnly** and **PreIndexAll**. For *median Count*, DIVA’s delay is $68.3\times$ shorter than the others. For *max Count*, DIVA spends 34 seconds on average ($635\times$ realtime), which is $5.8\times$, $5.0\times$, and $1.3\times$ shorter than **CloudOnly**, **PreIndexAll**, and **OptOp**.

Query progress DIVA makes much faster progress in most time of query execution. It *always* outperforms **CloudOnly** and **OptOp** during *Retrieval/Tagging* (Figure 4.7). It *always* outperforms alternatives in median/average count (Figure 4.8).

Why DIVA outperforms the alternatives? The alternatives suffer from the following.

- Inaccurate indexes. **PreIndexAll** resorts to inaccurate indexes (YOLOv3-tiny) built at capture time. Misled by them, *Retrieval* and *Tagging* upload too much garbage; *Counting* includes significant errors in the initial estimation, slowing down convergence.
- Lack of long-term knowledge. **OptOp**’s operators are either slower or less accurate than DIVA, as illustrated in Figure 4.4.

Table 4.5. DIVA’s performance (speedup) with various bandwidths. Numbers: min/median/max of times (\times) of query delay reduction compared to baselines (rows). Averaged on all videos and 9 bandwidths in 0.1MB/s–10MB/s.

	<i>Retrieval</i>	<i>Tagging</i>	<i>Count/Max</i>	<i>Count/Avg&Med</i>
CloudOnly	4.5/14.9/52.2	3.61/3.9/5.1	2.8/21.1/42.5	6.9/83.4/439.2
OptOp	2.2/4.1/4.9	2.0/2.3/2.6	1.2/1.5/2.1	6.9/83.4/439.2*
PreIndexAll	1.9/3.8/11.6	3.2/3.6/4.9	1.2/8.9/18.2	2.5/14.0/41.3

*: Fall back to CloudOnly as the camera does not execute NN for these query types

- One operator does not fit an entire query. Optimal at some point (e.g., 99% Retrieval), the operator runs too slow on easy frames which could have been done by cheaper operators.

Why DIVA underperforms (occasionally)? On short occasions, DIVA may underperform PreIndexAll at early query stages, e.g., BoatHouse in Figure 4.7. Reasons: (1) PreIndexAll’s inaccurate indexes may be correct on *easy* frames; (2) PreIndexAll does not pay for operator bootstrapping as DIVA. Nevertheless, PreIndexAll’s advantage is transient. As easy frames are exhausted, indexes make more mistakes on the remaining frames and hence slow down the query.

Can DIVA outperform under different network bandwidths at query time? Table 4.5 summarizes DIVA’s query delays at 9 bandwidths evenly spaced in [0.1 MB/s, 10 MB/s] which cover typical WiFi bandwidths [25]. We have observed that: on lower bandwidths, DIVA’s advantages over baselines are more significant; at high bandwidths, DIVA’s advantages are still substantial ($>2\times$ in most cases) yet less pronounced. The limitation is not in DIVA’s design but rather its current NNs: we find it difficult to train operators that are both fast enough to keep up with higher upload bandwidth and accurate enough to increase the uploaded positive ratio proportionally.

vs. “all streaming”: network bandwidth saving Compared to streaming all videos (720P 1FPS) at capture time, DIVA saves traffic significantly, as shown in Figure 4.9. When only as few as 0.005% of video is queried as in our case study (Section 4.1), the saving is over three orders of magnitude. Even if all captured videos are queried, DIVA saves more than $10\times$, as its on-camera operators skip uploading many frames. Among the bandwidth

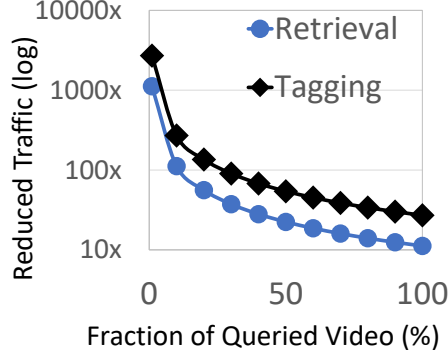


Figure 4.9. DIVA significantly reduces network traffic compared to “all streaming”. Results averaged over all videos.

reduction brought by DIVA, only less than 30% attributes to the background subtraction technique. It shows that the disadvantage of “all streaming” is fundamental: streaming optimizations may help save the bandwidth (upmost several times [18]) but cannot offset the waste, as discussed in Section 4.1.3.

DIVA vs. “all streaming”: query speed As “all streaming” uploads all videos to the cloud before a query starts, the query speed is bound by cloud GPUs but not network bandwidth. With our default experiment setting (1 GPU and 1MB/s network bandwidth), “all streaming” still runs queries much slower than zero streaming. Adding more cloud GPUs will eventually make “all streaming” run faster than DIVA.

Training & shipping operators For each query, DIVA trains ~ 40 operators, of which ~ 10 are on the Pareto frontier. The camera switches among 4–8 operators, which run at diverse speeds ($27\times$ – $1,000\times$ realtime) and accuracies. DIVA chooses very different operators for different queries. Training one operator typically takes 5–45 seconds on our test platform and requires 5k frames (for bootstrapping) to 15k frames (for stable accuracy). Operators’ sizes range from 0.2–15 MB. Sending an operator takes less than ten seconds. Only the delay in training and sending the first operator (≤ 40 seconds) adds to the query delay which is included in Figure 4.7–4.8. Subsequent operators are trained and transmitted in parallel to query execution. Their delays are hidden from users.

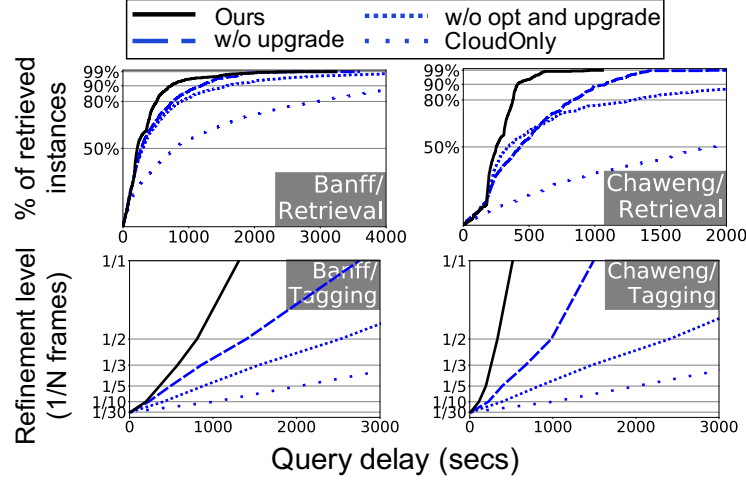


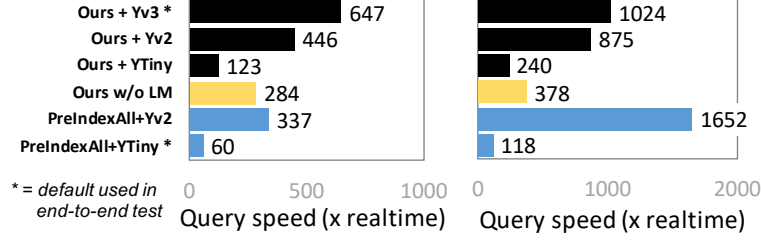
Figure 4.10. DIVA’s both key techniques – optimization with long-term video knowledge (opt) and operator upgrade (upgrade), contribute to performance significantly.

4.7.2 Validation of query execution design

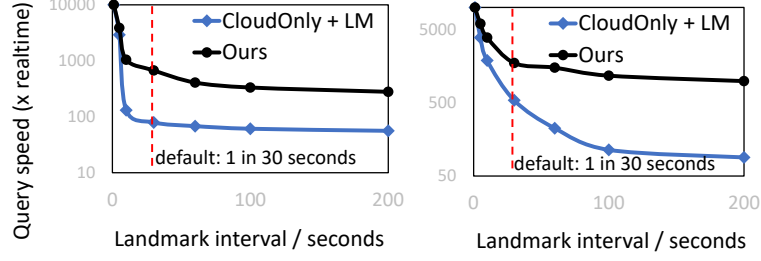
The experiments above show DIVA’s substantial advantage over `OptOp`, coming from a combination of two techniques – optimizing queries with long-term video knowledge (“*Long-term opt*”, Section 4.3) and operator upgrade (“*Upgrade*”, Section 4.4). We next break down the advantage by incrementally disabling the two techniques in DIVA. Figure 4.10 shows the results.

Both techniques contribute to significant performance. For instance, disabling *Upgrade* increases the delay of retrieving 90% instances by 2× and that of tagging 1/1 frames by 2×-3×. Further disabling *Long-term Opt* increases the delay of Retrieval by 1.3×-2.1× and that of tagging by 1.6×-3.1×. Both techniques disabled, DIVA still outperforms `CloudOnly` with its single non-optimized operator.

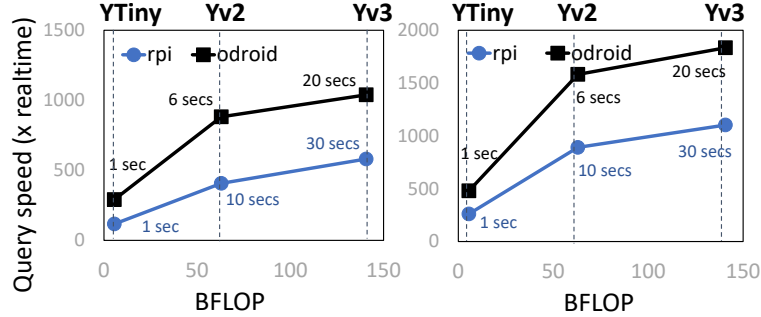
Upgrade’s benefit is universal; *Long-term opt*’s benefit is more dependent on queries, i.e., the skews of the queried object class in videos. For instance, DIVA’s benefit is more pronounced on Chaweng, where small bicycles only appear in a region in 1/8 size of the entire frame, than Ashland, where large trains take 4/5 of the frame. With stronger



(a) DIVA’s performance degrades significantly with less accurate landmarks (produced by Yv2 and YTiny), which can be even worse than no landmarks at all (“w/o LM”).



(b) DIVA’s performance degrades slowly with sparser landmarks. The y-axis is logarithmic.



(c) On given camera hardware (Rpi3/Odroid), sparser yet more accurate LMs always improve DIVA’s performance. Landmark intervals annotated along curves.

Figure 4.11. Validation of landmark design. In (a)/(b)/(c): Left – *Retrieval* on Chaweng; Right – *Tagging* on JacksonH.

skews in Chaweng, DIVA trains operators that are more accurate and run faster. This also accounts for DIVA’s varying (yet substantial) advantages over the alternatives (Figure 4.7).

Next, we deviate from the default landmark parameters (Table 4.4) to validate the choice of sparse-but-sure landmarks.

DIVA hinges on accurate landmarks. As shown in Figure 4.11(a), modestly inaccurate landmarks (as produced by YOLOv2; 48.1 mAP) increase delays for Q1/Q2 by 45%

and 17%. Even less accurate landmarks (by YOLOv3-tiny; 33.1 mAP) increase the delays significantly by $5.3\times$ and $4.3\times$. Perhaps surprisingly, such inaccurate landmarks can be worse than no landmarks at all (“w/o LM” in Figure 4.11): when a query starts, a camera randomly uploads unlabeled frames for the cloud to bootstrap operators.

Why inaccurate landmarks hurt so much? They (1) provide wrong training samples; (2) lead to incorrect observation of spatial skews which further mislead frame cropping; and (3) introduce large errors into initial statistics, making convergence harder.

DIVA tolerates longer landmark intervals. As shown in Figure 4.11(b), DIVA’s *Retrieval* and *Tagging* performance slowly degrade with longer intervals. Even with an infinite interval, i.e., “w/o LM” in Figure 4.11(a), the slowdown is no more than $3\times$. On *Counting*, the performance degradation is more pronounced: $5\times$ longer intervals for around $15\times$ slow down. Yet, such degradation is still much smaller than one from inaccurate landmarks (two orders of magnitude). The reason is that, with longer LM intervals DIVA has to upload additional frames in full resolution ($\sim 10\times$ larger than LMs) when a query starts for bootstrapping operators; such a one-time cost, however, is amortized over the full query.

Create the most accurate landmarks possible Should a camera build denser yet less accurate landmarks or sparser yet more accurate ones? Figure 4.11(c) suggests the latter is always preferred, because of DIVA’s high sensitivity to landmark accuracy and low sensitivity to long landmark intervals.

DIVA on wimpy/brawny cameras DIVA suits wimpy cameras that can only generate sparse landmarks. Some cameras may have DRAM smaller than a high-accuracy NN (e.g., ~ 1 GB for YOLOv3); fortunately, recent orthogonal efforts reduce NN sizes [155]. Wimpier cameras will further disadvantage the alternatives, e.g., **PreIndexAll** will produce even less accurate indexes. On higher-end cameras (a few hundred dollars each [156]) that DIVA is *not* designed for, DIVA still shows benefits, albeit not as pronounced. High-end cameras can afford more computation at capture time. i) They may run **PreIndexAll** with improved index accuracy. In Figure 4.11(a), running YOLOv2 on all captured frames (**PreIndexAll**+Yv2), DIVA’s performance gain is $1.9\times$ (left) or even $0.6\times$ (right). ii) These cameras may generate denser landmarks and rely on the cloud for the remaining frames. Figure 4.11(b) shows, with one landmark every 5 seconds, DIVA’s advantage is $1.5\times$.

4.8 Related work

Optimizing video analytics The CV community has studied video analytics for decades, e.g., for online training [157], [158] and active learning [159]. They mostly focus on improving analytics accuracy on short videos [5], [135], [136], [160]–[162] while missing opportunities in exploiting long-term knowledge (Section 4.3). These techniques alone cannot address the systems challenges we face, e.g., network limit or frame scheduling. A common theme of recent work is to trade accuracy for lower cost: VStore [18] does so for video storage; Pakha *et al.* [93] do so for network transport; Chameleon [6] and VideoStorm [9], [92] do so with video formats. DIVA’s operators as well exploit accuracy/cost tradeoffs. Multiple systems analyze archival videos on servers [7], [18], [99], [163], [164]. DIVA analyzes archival videos on *remote* cameras and embraces new techniques. ML model cascade is commonly used for processing a stream of frames [8], [165], [166]: in processing a frame, it keeps invoking a more expensive operator if the current operator has insufficient confidence. This technique, however, mismatches exploratory analytics, for which DIVA uses one operator to process many frames in one pass and produces inexact yet useful results for all of them.

Edge video analytics To reduce cloud/edge traffic, computation is partitioned, e.g., between cloud/edge [12], [35], [167], edge/drone [168], and edge/camera [13]. Elf [169] executes counting queries completely on cameras. Most work targets live analytics, processes frames in a streaming fashion and trains NNs ahead of time. DIVA spreads computation between cloud/cameras but takes a disparate design point (zero streaming) that are inadequate in prior systems. CloudSeg [36] reduces network traffic by uploading low-resolution frames and recovering them via super resolution. DIVA eliminates network traffic at capture time at all.

Online Query Processing Dated back in the 90s, online query processing allows users to see early results and control query execution [32], [170]. It is proven effective in large data analytics, such as MapReduce [133]. DIVA retrofits the idea for video queries and accordingly contributes new techniques, e.g., operator upgrade, to support the online fashion. DIVA could borrow UI designs from existing online query engines.

WAN Analytics To query geo-distributed data, recent proposals range from query placement to data placement [171]–[175]. JetStream [176] adjusts data quality to meet network bandwidth; AWStream [177] facilitates apps to systematically trade-off analytics accuracy for network bandwidth. Like them, DIVA adapts to network; unlike them, DIVA does so by changing operator upgrade plan, a unique aspect in video analytics. DIVA targets resource-constrained cameras, which are unaddressed in WAN analytics.

4.9 Conclusions

We propose zero streaming, shifting most compute from capture time to query time. We build DIVA, an analytics engine for querying cold videos on remote, low-cost cameras. At capture time, DIVA builds sparse but sure landmarks; at query time, it refines query results by continuously updating on-camera operators. Our evaluation of three types of queries shows that DIVA can run at more than 100× video realtime under typical wireless network and camera hardware.

5. CLIQUE: SPATIOTEMPORAL OBJECT RE-IDENTIFICATION AT THE CITY SCALE

5.1 Background & motivations

5.1.1 System model

Queries & videos We target retrospective queries: at the query time, all videos are already stored in a central repository. We assume a large repository of videos from geo-distributed cameras. Preprocessing at ingestion, i.e., as videos are being captured, is optional, as permitted by compute resource. At ingestion time, the system knows the object classes that may be queried, e.g., cars, humans, but not the input images of queries. However, ingestion preprocessing does not know about future queries and is agnostic to specific queries.

A query includes an input image of the target object X and the scope of videos to be queried; the query does not carry any metadata, e.g., a timestamp or the origin camera that produced the input image. Following the norm in ReID research [42], [44]–[48], we do not assume the video repository contains the input image; we do not assume any other images from the origin camera is available.

Camera Deployments We make minimum, qualitative assumptions on camera deployment. The deployment covers multiple geo-locations. At each location, multiple cameras are co-located as a *geo-group*. The query system knows which cameras are co-located, i.e., belonging to the same geo-group. Of the same geo-group and during a short period of time, e.g., tens of seconds, cameras are likely (although not necessarily) to capture similar sets of objects from different viewpoints. Such kind of deployment can be easily observed in public dataset [37] and real-world applications [38], which we believe is the future trend of surveillance camera deployment.

We do not assume that the query system knows quantitative camera postures and quantitative correlations across camera geo-groups (e.g., “one object appearing in geo-group A has 50% chance to reappear in geo-group B within the next 10 minutes”). There is no sufficient evidence showing such knowledge is standard in city camera deployment. As such, we design

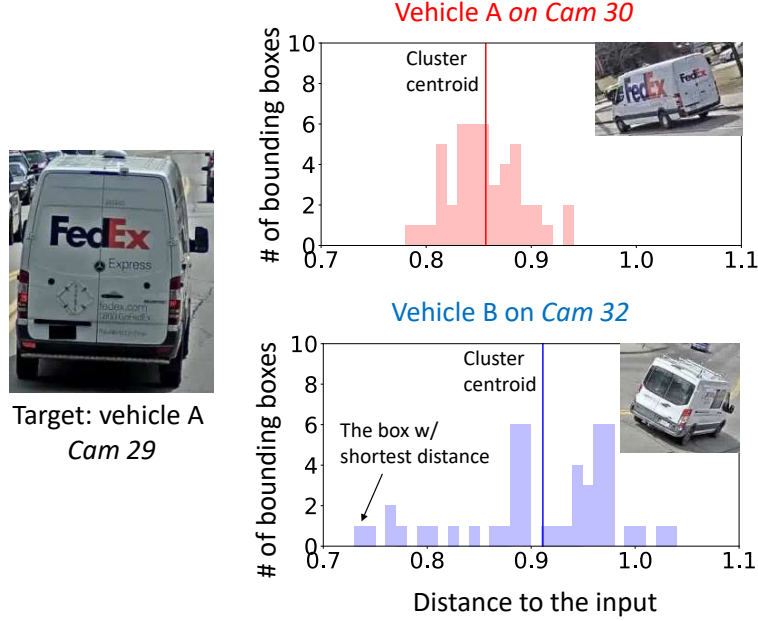


Figure 5.1. Examples of unreliable bounding boxes. (Left) an image of vehicle A, whose feature is the input. (Top) a histogram of distances between the input and other features of A. (Bottom) a histogram of distances between the input and features of B, a confusing vehicle. All features are 1×1024 vectors extracted by ResNet-152. Euclidean distances with L-2 norm [51] are used. Video clips: 4.7/4.9 sec for vehicle A/B from CityFlow [37].

a more generic system without such knowledge, and evaluate how the resultant system can be augmented in case some knowledge, e.g., camera correlations and orientations, becomes available (Section 5.6).

5.1.2 Challenge 1: Algorithm limitations

Observation: unreliable bounding boxes Figure 5.1 compares the features of a target vehicle A and a confusing vehicle B. The features are extracted by ResNet-152, a state-of-the-art neural network. Given an image of vehicle A: (1) ResNet-152 deems 10% of B’s bounding boxes exhibit shorter feature distances than A’s, hence are more similar to the input image, as compared to A’s other bounding boxes; (2) the bounding box closest to the input image is from the confusing vehicle B but not A; (3) Bounding boxes of the same vehicle show a high variation, as reflected by the wide range of the feature distances. The above example of unreliable bounding boxes is not isolated: they are the major hurdle for

ReID accuracy, responsible for an average of 0.65 loss of accuracy in 20% of queries executed by a baseline design (Section 5.6).

The causes Why unreliable bounding boxes? We explain the root cause with a simple formula:

$$Cam'(X) = Cam(X) + \mathcal{N} \quad (5.1)$$

Equation 5.1 describes how $Cam'(X)$, a camera’s actual observation of an object, is formed. X is the object’s inherent characteristics, e.g., its color, shape, skeleton, and key points. Two factors prevent a ReID system from directly learning X and matching it to the input. First, a camera’s posture modulates X as $Cam(X)$, i.e., the camera’s *ideal* observation on X . Second, the ideal observation is susceptible to transient disturbances \mathcal{N} , e.g., changes in resolution and viewpoint as objects move, background clutter, and occlusion. The impacts of camera postures and transient disturbances are strong, sometimes even stronger than the impact of inherent characteristics X . We have observed a different camera viewpoint of the same object resulting in $3\times$ difference in feature distances. Hence, classic ReID pipelines that aim at labeling each bounding box are in fact deciding on $Cam'(X)$, which encodes the camera posture and also the disturbances. The pipelines cannot achieve high accuracy because it is difficult to model $Cam()$ and \mathcal{N} and eliminate their impacts accordingly.

5.1.3 Challenge 2: Numerous cameras & videos

Colossal data volume A city camera generates more than 6 GBs of videos daily (720P at 1FPS). Estimated from recent reports on camera deployment in northern American cities [37], [38], [53], the number of city cameras per square mile ranges from a few hundred to a few thousand. A ReID query covering only a few square miles and one day of videos will have to consume PBs of videos.

Expensive pipelines Extensive work has been proposed to use neural networks (NNs) for ReID, advancing the accuracy steadily [178] on public datasets [179], [180]. For instance, recent pipelines cascade multiple NNs, each detecting a separate set of vehicle attributes, e.g., orientations and roof types. The additional NNs are reported to improve accuracy

(mAP) by 10% with up to $7\times$ overhead [46], [181]. We estimate that they can run no more than 15 FPS on a modern GPU.

Would cheaper features help? Cheaper NNs and vision primitives are unlikely remedies. The former were used by many *object detection* systems to provide a middle ground between high accuracy and low cost [8], [12], [15], [182]. On the much harder ReID tasks, however, modern NNs simply do not offer surplus accuracy for systems to trade off. We have tested RGB histogram [183] and SIFT [184], two cheap vision primitives for extracting object features. RGB features are highly volatile to lighting conditions and background clutter; SIFT yields poorer features compared to modern NNs, while not running significantly faster than the latter. Prepending these primitives to a ReID pipeline are likely to hurt performance.

5.1.4 Why is prior work inadequate

Computer vision research typically treats ReID as image retrieval [41], [46], [52], [181]. Aiming at finding all bounding boxes of a target object, computer vision studies typically focus on improving accuracy without considering query speed or efficiency much. Yet, retrieving every bounding box would miss opportunities, as we will show, that can provide useful spatiotemporal answers with much lower delays.

Existing ReID systems often consider smaller camera deployment and are evaluated on such datasets, e.g., 8 cameras over a university campus [179]. Many core designs depend on deployment-specific knowledge. For instance, ReXCam [56] searches in cameras among which spatial correlations are both strong and known. Given an input image captured by a *known* camera in the network, the system exploits camera correlation to find all images of the object. ViTrack [185] models and then predicts object trajectories in answering ReID queries. However, some assumptions (e.g., camera correlations) do not necessarily hold at the city scale; some others (e.g., a known origin camera) restrict use cases and are incompatible with the norm in ReID research [42], [44]–[48]. Without such strong assumptions, we intend our base design to be generic, and can nevertheless optimize queries with additional information as they become available (Section 5.6).

Spatiotemporal databases are designed for managing object trajectories, e.g., airplane movements and human movement, and answering queries on them [186]–[190]. They ingest *structured* data, e.g., sequences of time-location tuples as from GPS; they cannot ingest unstructured video data and recognizes object occurrences as Clique does. The output of our system can be the source of a spatiotemporal database.

5.2 Clique overview

Video ingestion At ingestion time, Clique optionally pre-processes videos from a small number of cameras, as permitted by available compute resource. The pre-processing detects objects of interesting classes (e.g., cars) and extracts their features; it is agnostic to specific queries. The pre-processing is elastic; Clique will run unfinished pre-processing at the beginning of a query’s execution.

Clique runs profiling as it ingests videos, a common practice of video systems [15], [56], [182]. It periodically samples videos from each camera to train several parameters used in clustering bounding boxes and determining camera sampling order. We will discuss these parameters in detail in Section 5.3 and 5.4. The profiling is light, processing 30 seconds of every 1-hour video and taking less than 10 seconds on a modern GPU.

Executing a query Clique organizes all the queried video footage by cells, each consisting of video clips captured near a location during a fixed period, as shown in Figure 5.2.

To execute a query, Clique searches in all cells iteratively; it adds cameras to each cell for processing in an incremental fashion. It starts by sampling from all cells in the query scope. The initial sampling is brief, as it only processes a small fraction of video footage in each cell – from selected cameras (“starter cameras”) ❶. From the sampled video of a cell, Clique detects distinct objects out of unreliable bounding boxes; it does so by clustering features of bounding boxes by similarity. Clique treats each resultant cluster representing a distinct object, where the cluster’s centroid is an approximation of the object’s feature ❷. Clique ranks all the cells by their promises, estimated from similarity between their enclosed objects and the input image ❸. Clique emits the ranked cells as query results to the user, who reviews the top ones ❹. Clique selects additional cameras for processing and

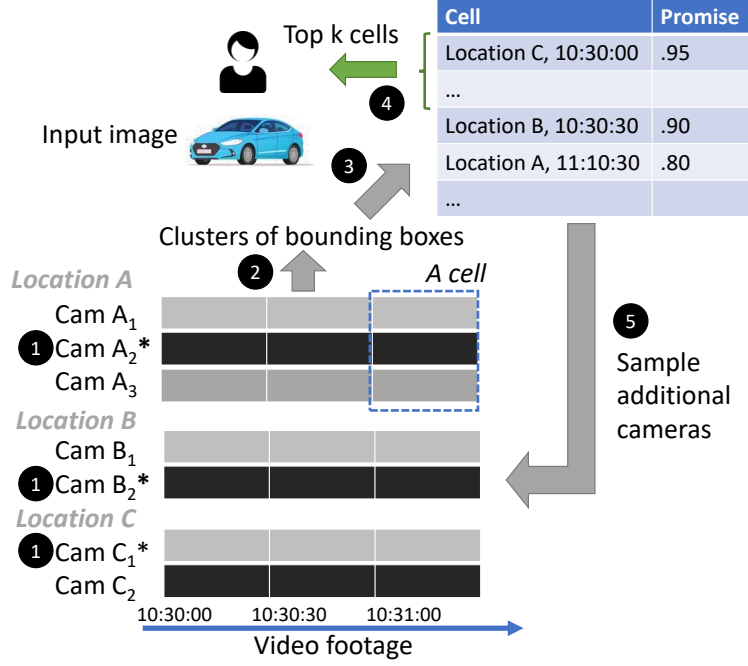


Figure 5.2. An overview of Clique. * = a starter camera.

uses the results to update the cell list continuously ⑤. A query is terminated by the user manually (e.g., when she is satisfied with results) or when Clique finishes processing videos in all queried cells.

Limitations Clique inherits the statistical nature of its underpinning ReID algorithms, notably the neural networks. While Clique empirically shows high confidence in its query results, e.g., it finds all true cells in more than 70% of queries (Section 5.6), it, however, cannot provide sound guarantee to do so. Similarly, although Clique’s accuracy often quickly converges during query execution, there is no guarantee on the convergence rate, e.g., processing 50% of videos to reach accuracy of 0.75. The hope is that users review the top k cells for true results; they entrust Clique on the remaining, unreviewed videos, being comfortable with the level of confidence that Clique provides. However, in case they want to be absolutely certain that no true cells are left out, they would need to inspect all the videos.

5.3 Clustering unreliable bounding boxes

A core mechanism of Clique is to recognize distinct objects from bounding boxes and compare the recognized objects to the input image. It addresses two concerns:

- Working around unreliable features of bounding boxes resulted from the limitations of ReID algorithms.
- Tolerating low frame rate, which allows Clique to sample more cameras, one of our principles in Section 1.3.

Observations on transient disturbance How to match an input image to numerous unreliable bounding boxes, each encoding impacts of transient disturbance? The disturbance is time-varying and its impacts can be either graduate or sudden. For example, as a vehicle travels through a camera’s field of view, its bounding box may resize; its view angle may change; occasionally, it may be occluded by a light pole; its background may be intruded by another vehicle.

Key idea: clustering similar bounding boxes Disturbance to bounding box features is difficult to model and eliminate in general. Yet, if we consider similar bounding boxes in consecutive video frames, their distorted features due to graduate impacts are likely to smooth out, and the outlier features due to sudden impacts can be removed from consideration.

To this end, Clique clusters object features based on their similarities. The similarities, for instance, can be measured by Euclidean distances across 1024-dimension feature vectors. As a result, each cluster represents a camera’s *general impression* on a distinct object during a given time window. The cluster’s centroid is an approximation of the camera’s ideal observation of the object.

Clique’s use of clustering is novel, in that it overcomes the accuracy limitations on individual bounding boxes. Notably, it differs from prior video systems [15] that cluster objects for efficiency, e.g. to avoid processing similar objects in a cluster.

Figure 5.1 showcases why clustering is useful. Recall that this example shows the difficulty in comparing an input image to individual bounding boxes (Section 5.1). However, once we

cluster the respective bounding boxes of the two vehicles, the centroids (distances as solid vertical lines) are much more robust indicators of object similarity, suggesting that the general impression of vehicle A is much closer to the input image.

In practice, we find simple clustering algorithms often suffice. Clique runs k-means clustering [61] within each spatiotemporal cell. By minimizing the sum of intra-cluster variances across all clusters, k-means thus effectively puts most visually similar objects in the same cluster. k-means guarantees convergence to local optimum and is known robust to outliers [62]. We also tested other popular clustering methods, e.g., hierarchical clustering [63]. We find them less favorable than k-means, e.g., they often attribute bounding boxes of the same object to separate clusters.

Predicting the number of distinct objects (k) As a prerequisite for applying k-means in a video clip, Clique must specify k as the number of distinct objects in that video. An accurate k is crucial to the clustering outcome.

Clique predicts k based on a simple intuition: of a given video scene, the distinct vehicle number is correlated to the spatial density of bounding boxes. Therefore, Clique only needs twofold information to predict k : (1) x_1 , the number of bounding boxes detected in the video clip; (2) x_2 , the number of frames that contain non-zero objects. Such information is already available from object detection, the ReID stage that precedes feature extraction described in Section 1.3. From x_1 and x_2 , Clique further derives three variables as different orders of the box/frame ratio: $x_3 = (x_1/x_2)^2$, $x_4 = (x_1/x_2)$, and $x_5 = (x_2/x_1)$.

We formulate classic kernel ridge regression [191]: $k = \mathbf{a}\mathbf{x} + b$. The model takes as an input $\mathbf{x} = [x_1, x_2, x_3, x_4, x_5]$ which consists of all aforementioned variables; its parameters are a vector \mathbf{a} and a scalar b . We instantiate one model for all the cells, for which we train a and b offline in one shot on 30-second labeled videos from 25 cameras.

Tolerance of low frame rates k-means clustering is robust to low frame rates, suiting our design principle stated in Section 1.3. As a comparison, we have investigated object tracking, another well-known approach to differentiating objects [192], [193]: first detecting individual bounding boxes on all frames; then linking bounding boxes across consecutive frames as distinct objects based on estimation of their motion trajectories. Yet, to estimate trajectory with good accuracy, object tracking demands a much higher frame rate than

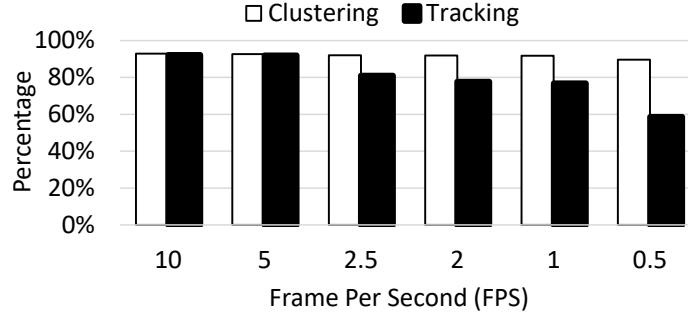


Figure 5.3. Clustering of bounding boxes tolerates low frame rates. Y-axis: the percentage of bounding boxes correctly attributed to respective objects. Object tracking implemented in OpenCV 3.4.4. Videos from CityFlow [37].

clustering. Figure 5.3 shows an experiment: while both k-means and object tracking can identify distinct objects well with high frame rates, e.g., 10 FPS, as the frame rate drops to 2.5 FPS or lower, object tracking quickly loses accuracy to be barely useful (~ 0.6 with 0.5 FPS). By contrast, k-means still maintains a high accuracy over 90%. Object tracking also suffers from other difficulties, e.g., differentiating multiple nearby objects following similar trajectories [194], [195].

Increasing frame rate for clustering, on the other hand, leads to a diminishing return: the accuracy improves by less than 3% by increasing from 1 to 10 FPS. This supports our principle: prioritizing camera coverage over video quality.

5.4 Incremental search in spatiotemporal cells

Clique’s search mechanism addresses two design questions: (1) how likely does a cell contain the target object; (2) for which cells Clique should process additional cameras. The former question determines the order of Clique processing undecided cells and the order of Clique presenting decided cells to users for review. The latter question guides Clique’s search direction.

5.4.1 Assessing cell promises

Clique estimates how likely a cell contains the target object by *promise*. The rationale is that a cell shows high promise as long as any object in this cell is highly similar to the

input image. Based on this rationale, we define the *single-camera promise*, $p_{single}(R, \mathcal{C})$, as the promise Clique would perceive in cell \mathcal{C} by only processing a video clip from a camera R : it is *reciprocal* to the smallest feature distance between the input and any centroid of object clusters from the video. That is, $p_{single}(R, \mathcal{C}) = 1/\min(\text{dist}(X, o))$ where $o \in \text{objs}$ and X is the feature of target object. As Clique processes video clips from additional cameras for a cell \mathcal{C} , it estimates the cell’s overall promise, i.e., multi-camera promise, as the highest of single-camera promises of \mathcal{C} .

The promise metric reflects our intuition: a cell appears more promising to Clique (with a higher multi-camera promise) as long as the cell has one strong champion camera (showing high single-camera promise) than having multiple weak supporters (medium single-camera promises).

5.4.2 Prioritizing cells in search

A cell’s promise reflects the single most similar object recognized in a cell. It, however, is inadequate for Clique to decide whether a cell is worth further exploring, i.e., to process more cameras for the cell. To do so, Clique needs to track the *accumulated* evidence discovered in the cell and the *accumulated* search efforts spent on the cell so far. Neither is reflected in the cell’s multi-camera promise. For instance, if Clique only stops processing additional camera for cells with high enough promise, it may process too many cameras for cells where no single object is highly similar to the input.

To this end, Clique puts all the cells in three categories:

- The green cells: Clique has collected enough evidences – though not necessarily all – for them, and predicts them *likely* to contain the target object. Processing additional cameras is unlikely to change this assessment.
- The red cells: Clique has collected enough evidences and predicts them *unlikely* to contain the target object. Processing additional cameras is unlikely to change this assessment.

- The gray cells: the existing evidences are insufficient. Processing one or a few cameras will likely turn the cells to red or green. This is how a human analyst would make up mind on a suspicious cell – by inspecting additional video footage from a different camera viewpoint.

Search plan All cells will begin in gray. Clique navigates its search from the gray cells (to resolve the undecided cells), to the green cells (to refine the order in which they will be presented to the user), and then to the red cells (in the unlikely event of any true cells are left out). Based on new processing results in a cell, Clique updates its category accordingly as will be discussed below. Clique will exhaust processing cells in a category before moving to the next category. In each category, it always processes the cell that has the highest multi-camera promise.

Categorizing a cell with voting To determine the category for a cell \mathcal{C} , Clique runs a simple voting mechanism to incorporate observations of multiple cameras. The voting mimics how humans would make decision out of a set of expert opinions. Clique quantizes all single-camera promises with two thresholds, P_{high} and P_{low} . To \mathcal{C} , a camera with promise p casts a high-confidence vote with a weight of 1 if $p > P_{high}$; it casts a medium-confidence vote with a weight of $1/k$ if $P_{high} < p < P_{low}$. The resultant vote count is more intuitive and tunable than, e.g., a sum of numerical single-camera promises. By tuning k , we control the relative weights of high/medium confidence votes. In the current implementation, we sets $k=2$. That is, Clique moves a cell to the green as long as Clique has collected two medium-confidence votes for it.

The threshold parameters P_{high} and P_{low} hinge on the tradeoff between refining existing results and exploring for new results. A lower P_{high} would eagerly put cells in the green category, postponing processing additional cameras for them until much later in the search process. By comparison, a higher P_{high} would be more reluctant in turning cells green; Clique will only pause processing on them if evidence is strong.

Given that Clique is a recall-oriented system to minimize human effort, we set P_{high} high so that Clique continues spending resource on promising cells to refine their ranking. This is because we expect users to only inspect the top few cells; it is thus vital to include true cells

in this small range. Based on the same rationale, we tune P_{low} to a low value to admit more cells to the gray category. We set $P_{high} = 1/d_{short}$, where 99% of the bounding boxes with feature distance shorter than d_{short} belong to the same vehicle. We set $P_{low} = 1/d_{long}$, where 99% of the bounding boxes that belong to the same vehicle have feature distances shorter than d_{long} . We will evaluate their sensitivity.

5.4.3 The search process

Stage 1: Initial sampling of cells Clique starts a query by sampling from all cells and processing one camera for each.

Based on videos from the starter cameras, Clique recognizes distinct objects in each cell; for each recognized object, Clique derives their cluster centroids. Clique may prioritize cells if heuristics is available on which cells are more likely to contain the target object, e.g., from rush hours or busier traffic intersections. As Clique uses a low video frame rate (1 FPS) tolerable to the clustering algorithm (Section 5.3), it cover starter cameras from all cells with the lowest total cost. As initial sampling is done without information of the input image, it can be executed at the ingestion, as will be discussed below.

After processing the starter cameras for all cells in the query scope, Clique has the initial categories of cells with cells in each category ranked by their promises.

Choosing starter cameras The choices matter as they set the initial direction for search. Ideally, they should be the cameras most likely to have captured the target from a viewpoint similar to the input image. In practice, one could exploit knowledge on camera deployment to help pick starter cameras, e.g., by choosing the camera that has the most similar viewpoint with the input image. Without assuming such a priori, our base design follows simple heuristics: picking cameras that has the highest density of distinct objects. The hope is that their chances of having captured the target are higher; if the target is captured, even from a different viewpoint than the input, the resultant bounding boxes would show a decent similarity to the input and thus a high promise to Clique. Clique profiles each camera’s density of objects offline and picks the starter cameras ahead of query. We evaluate sensitivity of starter camera choices in Section 5.6.5.

Initial sampling at ingestion time Independent of input, the initial stage can be executed before queries. Pre-processing at ingestion is optional and elastic. The number of starter cameras Clique can process depends on resources, e.g., the number of GPUs owned by Clique. Clique processes unprocessed starter cameras when a query starts, and caches the results for subsequent queries on the same scope of videos (Section 5.5).

We also consider a situation of ample resources available to ingestion. Is it worth pre-processing multiple starter cameras per geo-group? Our experiments, as will be shown in Section 5.6, suggest diminishing returns. This is because a small number of starter cameras properly chosen can yield sufficiently accurate cell promises and the initial ranking.

Stage 2: Incremental search Based on initial sampling of starter cameras, Clique may be undecided on putting on a cell in the green or red category: the starter may completely miss the target vehicle; its viewpoint on the target vehicle may differ significantly from the input image; or the starter may have just captured a different, but visually similar vehicle.

Specifically, Clique picks the next cell as follows: if the highest ranked gray cell that still has unprocessed cameras, Clique processes one additional camera for it; if such gray cells are already exhausted, Clique processes the highest ranked green cell that still has unprocessed cameras; if no such cells, Clique moves to red cells, in hope of finding target object instances in those cells missed out previously. After selecting the next cell and processing one additional camera for it, Clique updates the cell’s category and re-rank the cells. The updated categories and ranks will be Clique’s basis for picking the next cell.

5.5 Optimizations

5.5.1 Optimizations with extra knowledge

We present the following add-on optimizations. They are based on assumptions that we intentionally left out from Clique’s base design. We will evaluate them in Section 5.6.7.

Picking starter cameras based on posture similarity If the quantitative postures of deployed cameras are known to Clique, e.g., as part of per camera metadata, Clique can pick starter cameras as ones having the most similar postures to the origin camera. The rationale is that if the target object is captured by starter cameras, a similar viewpoint will

boost the camera’s confidence. To do so, Clique needs to estimate the posture of origin camera, which is different in each query. On one hand, Clique may rely on human analyst to annotate the posture (one image per query); on the other hand, it may automate the estimation with vision operators proposed by active vision research.

Sampling cameras with complementary postures In its base design, when Clique samples a secondary (or subsequent) camera for a cell, it picks a random one from the same geo-group. Such decisions can be more informed by camera postures. While still keeping the choices of starter cameras, Clique picks the next camera as the one that offers the most different viewpoint compared to the prior camera sampled for, i.e., the N -th camera is always the camera that has the largest viewpoint difference with the $(N-1)$ -th camera.

Reusing states of previous queries Clique speeds up a query’s execution by reusing the states from prior queries on the same videos. These queries could be fully or partially executed. Their states include all distinct objects and their features from the starter cameras, and some of bounding boxes, distinct objects, and features from the remaining cameras. To the end, Clique reuses the existing distinct objects as the “free” estimation of the initial cell ranking; in incremental search, Clique may favor cameras for which partial results already exist. We will evaluate the former idea experimentally.

5.5.2 Utilizing cheap vision operators

We present the following possible optimizations from cheaper vision operators. We will evaluate them in Section 5.6.8.

Using cheap operators during ingestion for early ranking Clique’s pre-processing at video ingestion is elastic. As compute resources are precious for queries, Clique falls back to cheaper operators when there aren’t sufficient resource for expensive feature extraction during video ingestion. With cheaper features, Clique is able to derive a rough rank of spatiotemporal cells given the input image to guide future incremental search in cells. Within a wide spectrum of resource budgets, there are multiple choices of pre-process, e.g., processing more starter cameras by cheaper operators or processing less starter cameras by more expensive operators.

Using cheaper operators as early filters Cheap vision operators are widely used in prior systems [5], [8], [15], [91] at early stages to reduce query workloads by only focusing on relevant video footage/objects. To fully utilize cheaper operators, Clique could first extract cheap object features, pre-cluster those cheaper features, remove vehicle instances from those unpromising clusters that have a low p , and only extract expensive features on the rest of vehicle instances. As some clusters are removed by cheap filters in earlier stage, Clique adjusts k accordingly to the number of surviving clusters. However, clustering cannot effectively work on mixed features from different vision operators. For example, features from RGB histogram and ResNet-152 have different dimensions; even with the same dimension, e.g., ResNet-50 and ResNet-152, the feature representations from different operators still have different nature. Thus, cheap operators can act as early stage filters by removing unpromising object instances to reduce the effort of later expensive operators; they cannot replace the effort of expensive operators by splitting the workload across different vehicle bounding boxes.

Directly replacing expensive operators We also study Clique’s behavior by directly replacing expensive vision operator with cheaper ones.

5.6 Evaluation

We answer the following questions in evaluation:

§5.6.2 Can Clique achieve good accuracy with low delays?

§5.6.3 Are the key designs useful?

§5.6.4 Does Clique outperform prior alternative designs?

§5.6.5 How is Clique sensitive to its parameters and query inputs?

§5.6.6 How beneficial is processing at ingestion time?

§5.6.7 How effective are Clique’s add-on optimizations?

§5.6.8 How does the utilization of cheaper vision operators impact Clique’s performance?

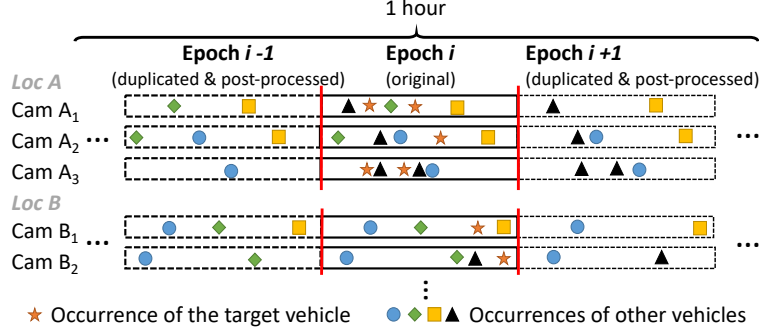


Figure 5.4. Augmenting real-world city videos [37] as our test dataset: duplicating the original epoch; erasing random vehicles from each epoch; erasing the target vehicle from all but the original epoch.

5.6.1 Methodology

Video Dataset An ideal video dataset for benchmarking Clique would: (1) consist of long videos produced by many cameras; (2) come from real-world deployment and capture spatiotemporal patterns of vehicles; (3) have vehicle labels as the ground truth for accuracy evaluation. Real-world videos are preferred over simulators, e.g., VisualRoad [196]: while simulators can generate long traffic animations with multiple viewpoints, the resultant bounding boxes are ultimately based on vehicle motions and camera postures specified by us; it is unclear how well they reflect ReID in the real world, e.g., rarity of target objects, diverse cameras, and transient disturbance.

Unaware of such datasets in public, we use CityFlow [37] published by NVIDIA for use in the AI City Challenge 2019. The dataset consists of 5 scenarios, from which we select the largest one (scenario 4). The scenario consists of 25 cameras at 7 traffic intersections (hence 7 geo-groups) of a northern American city. The scenario includes 30 minutes of videos, capturing 17,302 vehicle bounding boxes belonging to 70 distinct vehicles. We downsample videos to 1 FPS, a low frame rate adopted in prior video systems [8], [15], [18].

We overcome a key shortcoming of the CityFlow dataset: all videos are short, each lasting around 30 seconds on a camera. We therefore augment the dataset to extend video length. To do so, we make sure to: (1) preserve the vehicle spatiotemporal patterns, within and across camera geo-groups; (2) keep the bounding boxes of target vehicles rare. Our augmenting

Table 5.1. The augmented video dataset used in evaluation.

# of all cameras	25	Total video length	25 hours
# of geo-groups	7	# of distinct vehicles	70
# of total cells	3000	# of all bounding boxes	~1M
Time duration of a cell	30 secs		

procedure is shown in Figure 5.4. First, we extend each camera’s video by duplicating the original video clip as many “epochs”. Each epoch embraces videos clips from all cameras; within one epoch, the original spatiotemporal patterns are preserved. Second, we remove a random fraction (0–1) of vehicles from each epoch, erasing their bounding boxes from all the video clips in that epoch. This diversifies the augmented videos over time, preventing them from becoming repeated loops of the original clips.

We further ensure that target objects are difficult to find throughout all videos. For a query with an input image of target X, we exclude the origin camera from the query scope; we erase all X’s bounding boxes from duplicated epochs while only keeping ones in the original epoch.

The final videos used in evaluation are summarized in Table 5.1. It span 25 hours of videos, one hour per camera. Together, the videos consist of 3000 cells, each lasting 30 seconds; the videos include more than 1 million bounding boxes. Given a query, only 243 (0.02% of all) bounding boxes on average belong to the target vehicle, and 1.6 (0.5% of all) cells on average contain the target.

Query setup We test Clique on 70 queries, each for one distinct vehicle in the video dataset. A query contains one vehicle image randomly selected from all bounding boxes of the vehicle in the dataset. We then exclude the origin camera from the query scope. As described in Section 5.2, a query carries no metadata, e.g., the timestamp or the origin camera of the input image, as opposed to prior work [56], [57].

Environment Clique runs on a 12-core Xeon E5-2620 v3 workstation with a NVIDIA Titan V GPU. Clique runs YOLO [50] to detect vehicles and ResNet-152 [49] to extract features. We train ResNet-152 on images from 329 different vehicles from 34,760 images from CityFlow [37] and Cars [197] dataset, with all vehicles used in the evaluation excluded.

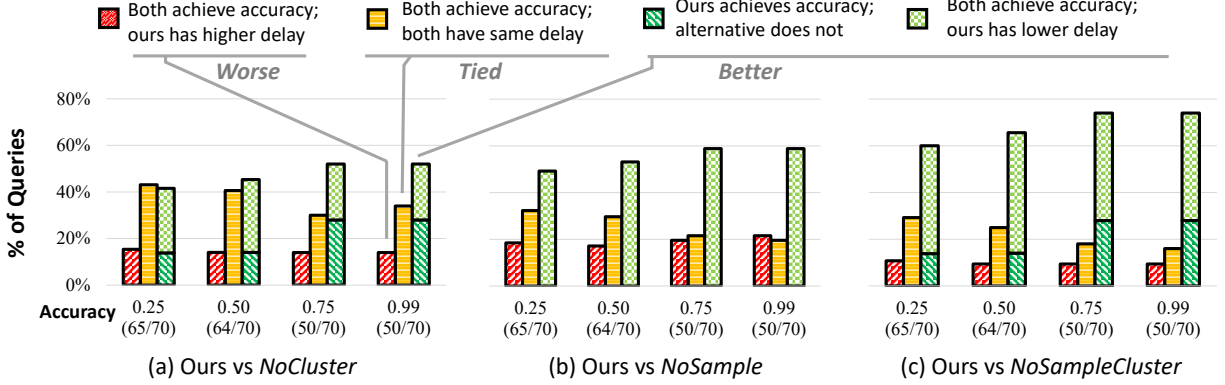


Figure 5.5. Query-by-query comparison between Clique and the alternatives, broken down by per-query comparison outcomes. Numbers on bottom: accuracy goals; (X/Y): X = number of queries that Clique reached the accuracy; Y = total query count.

Accuracy Metric We evaluate query accuracy with *recall at k*, i.e., the fraction of all true cells (i.e. containing the target object) that have been included in Clique’s top k output cells. *Recall at k* is commonly used for measuring accuracy of recall-oriented retrieval focusing on rare positives [59], [60]. By setting k as low as 5, the resultant metric (*recall at 5*) measures the usefulness of query results when used with low human efforts, i.e. when a user reviews the top 5 cells returned by Clique. A high value of *recall at 5* means that Clique successfully returns most if not all true cells to the user, since the true cells of most queries (> 98% in our dataset) are fewer than 5.

Speed metric As Clique keeps refining the rank of cells, we report the times since a query’s start until the output accuracy reaches a set of accuracy goals: 0.25, 0.50, 0.75, and 0.99.

5.6.2 End-to-end performance

Accuracy Clique achieves high accuracy for most queries. All 70 queries achieve an average recall at 5 of 0.87. Among them, 64 queries (91%) meet or exceed an accuracy of 0.50; 50 queries (70%) meet or exceed an accuracy of 0.99. Besides, all 70 queries achieve an average recall at 10 of 0.91. Among them, 69 queries (99%) meet or exceed an accuracy of 0.50; 58 queries (83%) meet or exceed an accuracy of 0.99. Such accuracy is higher

than what can be achieved on individual bounding boxes, as will be shown in Section 5.3. This validates our clustering approach: Clique can make robust decisions on cells based on unreliable bounding boxes.

We manually inspect the six queries where accuracy is low (< 0.5), attributing the root cause as the limitation of today’s feature extractors. For instance, for a query with input vehicle 262, it is challenging even for humans to associate the input image with all the 30 true bounding boxes; not surprisingly, their features show long distances to the input.

Delays Clique achieves accuracy goals with moderate delays. In querying 25 hours of videos on our single-GPU machine, Clique takes 59.5 seconds on average (stddev: 156.2, 90% percentile: 457.5) to reach an accuracy of 0.50, and 108.5 seconds on average (stddev: 194.9, 90% percentile: 488.0) to reach 0.99. Roughly, this speed is $830\times$ of video realtime, i.e., 4.3 seconds to perform ReID on each hour of videos.

5.6.3 Validation of key designs

Alternatives We compare Clique to the following alternatives.

- *NoCluster*: Clustering is turned off. The alternative ranks a cell based on the minimum pairwise distance between the input image and bounding boxes in the cell. With the rank, it searches in cells by sampling from cameras as Clique does.
- *NoSample*: Camera sampling is turned off. The alternative randomly picks starter cameras for each camera group. It clusters bounding boxes and ranks cells accordingly, just as Clique does. Unlike Clique which adds one camera to a cell and updates the cell rank, the alternative processes all cameras (in random order) for a cell before updating the rank.
- *NoSampleCluster*: Both clustering and sampling are off. The alternative ranks a cell based on the minimum distance between the input image and bounding boxes; it processes all cameras for a cell before updating the cell rank.

Figure 5.5 summarizes Clique’s competitiveness against the alternatives. On most queries, Clique outperforms the alternatives, either reaching higher accuracy or the same accuracy

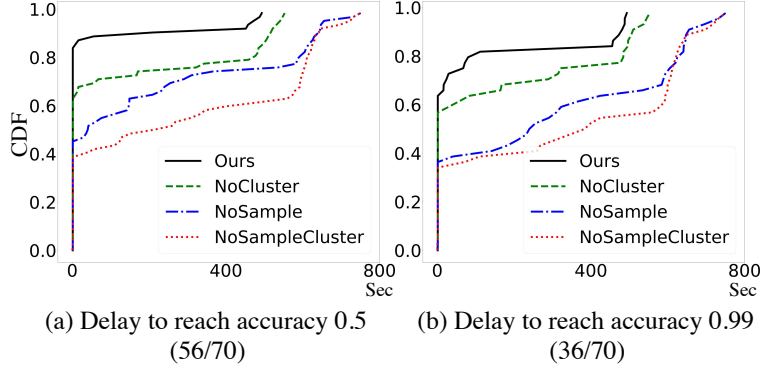


Figure 5.6. The CDF of query delays by Clique and the alternatives. (X/Y): X = the number of queries on which all the versions reach the accuracy goal; Y = total query count.

in lower delays. Only on a small fraction of queries Clique shows longer delays; Clique never fails to reach accuracy goals attainable to the alternatives.

Clustering improves query accuracy Clique’s *eventual* accuracy, i.e., the accuracy after processing all videos, reaches 0.87 averaged on all queries, while the alternatives (*NoCluster* and *NoSampleCluster*) reach 0.74 on average. The per-query accuracy gain is 0.13 on average (stddev: 0.28). Among all queries, Clique’s eventual accuracy is higher on 14 out of all 70 queries; on the remaining queries Clique’s accuracy ties with the above alternatives (mostly with short delays, see below) and is never lower. Clustering is vital in two ways: (1) it is robustness against outlier bounding boxes and strong disturbance, the key to achieve high accuracy goals such as 0.99; (2) based on clustering, Clique’s initial cell ranking is more accurate, ensuring speedy search.

Camera sampling reduces query delays We zoom in the queries and accuracy goals attainable to Clique and all the alternatives. Figure 5.6 shows the delay CDFs. With accuracy goals of 0.50 and 0.99, Clique’s delays are $2.9\times$ and $1.7\times$ shorter than *NoCluster* on average, $4.5\times$ and $3.3\times$ shorter than *NoSample* on average, and $6.5\times$ and $3.9\times$ shorter than *NoSampleCluster* on average. The alternatives suffer from poor starter cameras which in turn result in poor initial cell ranking.

5.6.4 Comparisons to alternative designs

We compare Clique with two prior solutions [56], [198] by borrowing their key ideas and adapt them to answer spatiotemporal vehicle ReID queries.

- **ReXCam-ST: Utilizing spatiotemporal correlations:** ReXCam-ST is augmented by ReXCam’s [56] spatiotemporal correlation model across cameras and answers spatiotemporal queries by ranking the cells accordingly. To do so, ReXCam-ST profiles the spatial correlation, i.e., the portion of overlapped objects across different cameras, and the temporal correlation, i.e., the time difference in which an object is likely to reappear in other cameras. At ingestion time, ReXCam-ST randomly processes one starter camera from each location and derives a rough rank as Clique performs. At query time, since ReXCam assumes known knowledge of target vehicle’s location/time information while Clique does not, ReXCam-ST identifies the current top cell as the starting point, and visits correlated cameras and time ranges to update the rank of cells accordingly.
- **PROVID-ST: Exploiting spatiotemporal constraints:** PROVID-ST borrows the idea from classic vehicle ReID pipelines [198], [199] that re-ranks the bounding boxes with spatiotemporal constraints, which verifies the possibility of whether a vehicle is possible to reappear in another location/time given the knowledge of its appearance in the current location/time. To compare with this line of work, we exploit the spatiotemporal similarity model proposed in PROVID [198] to answer spatiotemporal queries and re-rank the cells based on the spatiotemporal similarity. As computer vision pipelines typically assume no priority of processing all bounding boxes, PROVID-ST randomly processes the video footage from all cameras and time ranges. The calculation of PROVID’s spatiotemporal similarity is shown in Equation 5.2: $|D_i - D_j|$ stands for the physical distance between cell i and j ; D_{max} stands for the farthest distance between all camera groups; $|T_i - T_j|$ stands for the time difference between cell i and j ; T_{max}

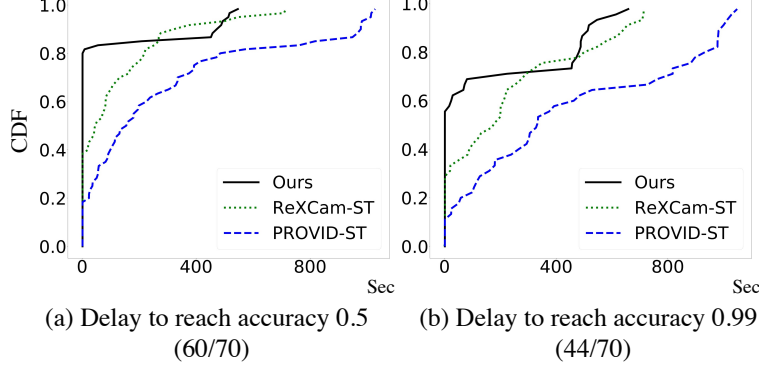


Figure 5.7. The CDF of query delays by Clique, ReXCam-ST, and PROVID-ST. (X/Y): X = the number of queries on which all the versions reach the accuracy goal; Y = total query count.

stands for the entire query time frame. A smaller s refers to a higher spatiotemporal similarity.

$$s = \frac{|D_i - D_j|}{D_{max}} \times \frac{|T_i - T_j|}{T_{max}} \quad (5.2)$$

To re-rank the spatiotemporal cells, we adjust promise p to be $p' = p/s$. As Clique does not assume any prior spatiotemporal information on the input image, to re-rank a cell i , the similarity s is derived against an average cell j averaged across all existing green cells.

Comparison results As shown in Figure 5.7, Clique outperforms ReXCam-ST and PROVID-ST. With accuracy goals of 0.50 and 0.99, Clique’s average delays are 38.3% and 32.7% shorter than ReXCam-ST, and $4.0\times$ and $2.8\times$ shorter than PROVID-ST, respectively. ReXCam-ST sometimes outperforms Clique as it can effectively narrow down the search scope into a short time range without processing unnecessary cells. Compared with Clique, PROVID-ST’s average accuracy drops by 13.8%, as the spatiotemporal similarity model proposed by PROVID [198] sometimes overlooks those cases when vehicles travel a long distance within a long time range, which typically end up with a low s based on Equation 5.2.

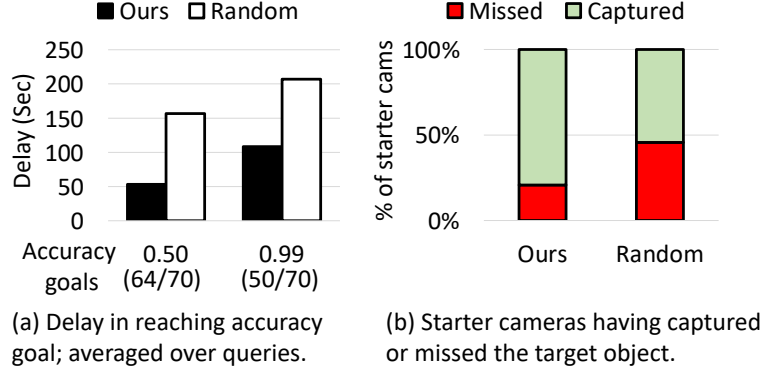


Figure 5.8. A comparison between Clique’s choices of starter cameras and random choices. (X/Y) in (a): X = number of queries that Clique reached the accuracy; Y = total query count.

5.6.5 Sensitivity to parameters and inputs

Choices of starter cameras matter While not affecting a query’s eventual accuracy, the choice of starter cameras has a high impact on query delays. This is because the choice affects the initial rank of cells. As shown in Figure 5.8(a), with starter cameras randomly picked, the average query delays grow by 1.5× and 3.9× to meet accuracy goals of 0.50 and 0.99, respectively. Figure 5.8(b) shows the cause: a substantial fraction of random starter cameras miss the target vehicle they should have captured (i.e., the target captured by other camera in the same geo-group), missing opportunity in optimizing the initial ranking of cells. With good choices of starter cameras, as shown in Figure 5.6, about 80% and 60% of queries can directly reach 0.50 and 0.99 without sampling additional cameras.

Moderate sensitivity to input images Clique shows resilience to different input images from our dataset. First, we replace the randomly selected input images with another random batch selected from the dataset. As a result, Clique sees an average of 0.04 difference in accuracy (stddev: 0.24); it sees delay differences of 7.7 seconds (11.7%) and 13.4 seconds (12.5%) for 0.50 and 0.99, respectively. Second, we test “easier” input images by including the camera that produced the input image in a query’s scope, while still excluding the input image from the scope. As such, the query scope now has a camera with a viewpoint identical to the input image. Clique sees moderate benefit: 0.05 higher accuracy on average (std: 0.15), 23.0% and 12.1% reduction in delays, and 1% less processed videos on average. We

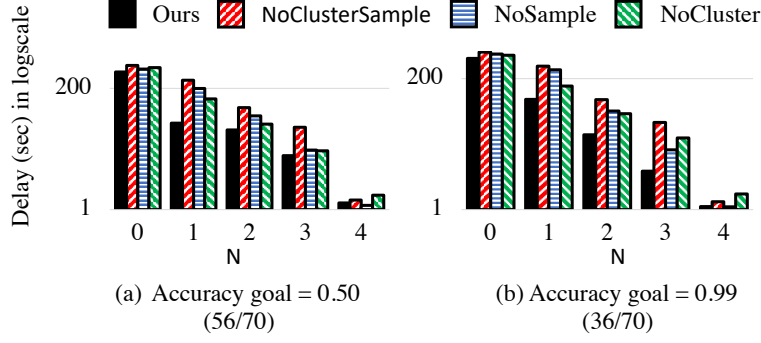


Figure 5.9. Query delays with N cameras per geo-group pre-processed at ingestion time. In case N exceeds the total cameras of a geo-group, all the cameras are pre-processed. (X/Y): X = number of queries on which **all** versions reach the accuracy goal; Y = the total query count. Y-axis in logscale.

attribute the results to our dataset characteristics: (1) Camera redundancy: given any input image, there are likely cameras offering similar viewpoints. (2) Decent image quality. Were the input images in poorer quality, e.g., with large occlusion or low resolution, they may confuse the neural networks used in Clique, resulting in lower accuracy overall.

Low sensitivity to thresholds We learn P_{high} and P_{low} via offline profiling using the original video clips. As the thresholds determine when to pause sampling cameras, their values may affect query delays but not the eventual accuracy. With methods described in (Section 5.4), we determine the default values as $P_{high}=1/d_{short}=1/0.73$ and $P_{low} = 1/d_{long} = 1/0.91$. We test Clique by deviating from such default values, i.e., $d_{short} \pm 0.1$ and $d_{long} \pm 0.1$. Across all 70 queries, the query delays only vary by less than 10% on average. The new thresholds increase delays on more than 90% of the queries (average increase: 2.4 seconds); and reduce delays on the remaining (average reduction: 7.2 seconds). Based on the minor variation, we conclude that the default parameters are adequate; the benefit from fine tuning thresholds for individual queries are marginal.

5.6.6 Delay reduction by processing at ingestion

Figure 5.9 shows average query delays with a variety of N , the number of starter cameras per geo-group pre-processed at ingestion time. With N exceeding 4 (not shown in the Figure),

Clique extracts all object features in real time, leaving only feature matching (negligible overhead) to query time.

The results support lightweight preprocessing at ingestion. (1) Pre-processing starter cameras reduces query delays substantially. Comparing to no pre-processing at all, pre-processing one starter camera per group reduces query delays by around $4\times$. (2) Pre-processing more than 1 starter camera per geo-group yields diminishing returns, no more than 25% delay reduction. (3) With the same pre-processing at ingestion time, Clique delivers much lower delays than the alternatives.

5.6.7 Impact of optimizations

We evaluate the deployment-dependent optimizations mentioned in Section 5.5.1.

Picking starter cameras based on orientations We estimate deployed camera orientations from the map provided by CityFlow [37] and use human-labeled viewpoints for input images of queries, minimizing inaccurate viewpoints. Overall, this optimization tends to benefit queries used to be slow, as shown in Figure 5.10. On the queries used to have $\geq 70\%$ percentile delays, Clique sees on average $5.8\times$ and $2\times$ lower delays in reaching accuracy of 0.50 and 0.99, respectively. For the remaining 70% queries, the delay reduction is negligible as most queries converge based on starter cameras. The reason is that a significant better viewpoints from manually picked starter cameras have little impact on current well-performed queries, but on those queries that used to suffers from bad indexes. Besides, by replacing starter cameras that used to have a decent viewpoint, the initial rank of spatiotemporal cells typically does not have sharp changes, so the query delay will not differ (those $< 70\%$ percentile).

Sampling cameras by complementary orientations With camera orientations of the dataset, Clique sees 4.6% and 2.0% reduction in delays to reach accuracy of 0.50 and 0.99, respectively. We identify two reasons for the minor benefit: co-located cameras are providing complementary viewpoints, and picking any of them is likely to help the search to similar degrees; the cameras with orientations opposed to the starter cameras can be inferior

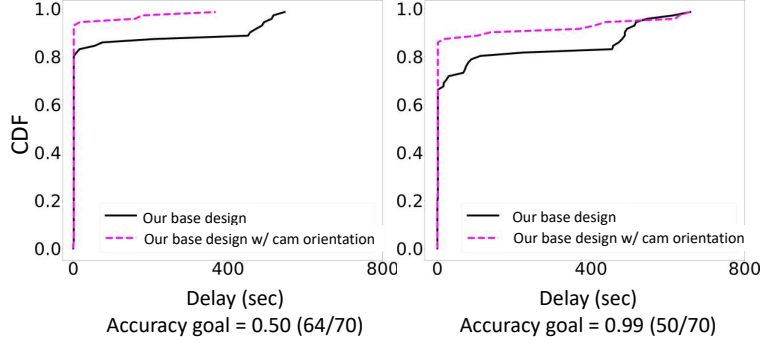


Figure 5.10. Delay CDFs of Clique augmented to exploit camera orientation knowledge. (X/Y): X = number of queries that Clique reached the accuracy; Y = total query count.

choices, e.g. capturing much fewer bounding boxes (and hence less likely the target object) than average cameras.

Reusing states of previous queries Clique can effectively speed up a new query by reusing intermediate state of previous queries. To show this, we test ten query pairs $\langle Q_{old}, Q_{new} \rangle$ on two accuracy goals of 0.50 and 0.99. The input images are randomly picked, and are different within each pair. Within each pair: we run Q_{old} , terminates it once reaching the accuracy goal, and run Q_{new} with the query state left from Q_{old} . Between pairs, we cleanse any query state. With Q_{old} reaching accuracy of 0.50 (i.e., a “brief” query), the delays for Q_{new} to reach accuracy of 0.50 and 0.99 are reduced by 86.2% and 76.8%, respectively. With Q_{old} reaching accuracy of 0.99 (i.e., a “thorough” query), the delays for Q_{new} are reduced by 86.2% and 78.1%, respectively.

5.6.8 Impact of cheaper vision operators

We test the designs that utilize cheaper vision operators mentioned in Section 5.5.2 with two extreme yet representative cheap vision operators, i.e., RGB histogram (a color filter) and ResNet-50 (an effective yet less reliable NN than ResNet-152). For Clique, we only consider global features rather than local features [184], as local features are typically used to perform pairwise matching between two images rather than clustering, and it is hard to concatenate multiple local features capturing diverse object details to a single global feature.

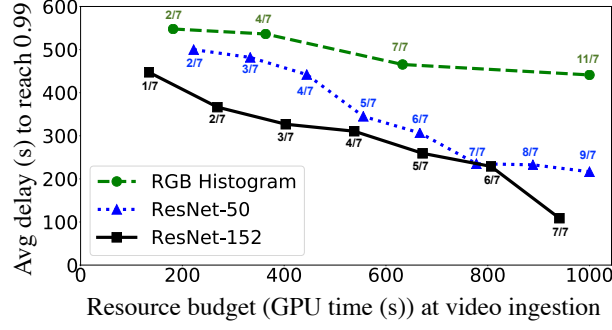


Figure 5.11. The average delay to reach 0.99 under different pre-processing budgets during ingestion. (X/Y): X = starter cameras covered; Y = total number of locations. X > Y means covering more than one starter cameras per location.

Using cheap operators during ingestion for early ranking Utilizing cheaper operators during video ingestion for early ranking is not substantially effective. As shown in Figure 5.11, the X-axis represents the spectrum of GPU time budget for pre-process: covering only one starter camera in all locations to one starter camera in each location (by YOLOv3 and ResNet-152). Under the same amount of resource budget, when Clique can afford covering substantial amount of locations by cheaper vision operators like RGB histogram or ResNet-50, Clique sees monotonic shorter delays. After covering all locations, adding more starter cameras in each location shows diminishing return, as shown by the green dashed line (RGB histogram) and the blue dotted line (ResNet-50). However, cheaper vision operators never beats ResNet-152 in the wide spectrum of ingestion budgets, and the results imply that investing resources in more expensive vision operators is always more rewarding.

Using cheap operators as early filters Cheap early filters results in lower accuracy and longer delays. By pre-clustering rough color histograms by RGB histogram, Clique only runs ResNet-152 on surviving vehicle instances, and the query accuracy drops 21%, while in the meantime incurs $3.4\times$ and $2.7\times$ longer delay to reach an accuracy of 0.50 and 0.99. Though significantly faster, color histograms are not reliable and many true vehicle instances are falsely removed after the pre-clustering. By pre-clustering more reliable features extracted by ResNet-50, the query accuracy drops 12%, while in the meantime incurs $2.7\times$ and $2.5\times$ longer delays to reach an accuracy of 0.50 and 0.99. Though deriving more reliable

features, ResNet-50 incurs non-negligible query delay as it is not significantly faster than ResNet-152.

Directly replacing expensive operators Cheap operators are less effective in ReID tasks. By replacing ResNet-152 with rough color filters like RGB histogram, Clique is unable to conduct effective vehicle re-identification, and the accuracy drops to nearly 0. By replacing ResNet-152 with another effective yet a little bit cheaper operator, ResNet-50, the accuracy drops 3.2%, and the delays to reach accuracy of 0.99 increased by 10%, as the features are less reliable and thus Clique’s dynamic rank of cells are less accurate to guide incremental searches.

5.7 Related work

We discuss related work not covered previously.

Optimizing video analytics Besides ReXCam [56] and ViTrack [185] discussed earlier, to reduce multi-camera inference cost, Caesar [200] encodes object activity correlation across cameras; Optasia [201] shares common work modules and parallelizes query plans; Jiang *et al.* [202] initiate an abstraction of camera clusters to enable resource/data sharing among cameras. There has been many works proposed to optimize video analytics with operator cascades [5], [8], [15], [91], and format tuning to trade accuracy for cost-efficiency [6], [9], [18], [92], [93]. Focus [15] saves cost by pre-processing videos with cheap NNs at ingestion. Notably, it clusters object features to avoid redundant comparisons with target objects. Clique uses clustering in a different way: to smooth out transient disturbances for higher ReID accuracy. Extensive works are proposed to exploit collaborations between cloud and edge [12], [35], [167], [203], [204]; cloud/edge and mobile devices [14], [205]; cloud and cameras [19]; edge and cameras [11], [13]; and edge and drones [168]. Elf [182] imposes energy planning for counting queries on resource-frugal cameras. None above was designed for ReID queries over city-scale cameras.

Information Retrieval Recall-oriented retrievals, e.g., legal or patent search, is a group of tasks to find all relevant documents and a bad ranking typically incurs significant search efforts from domain experts [59], [60], [206]. As objects are rare in ReID tasks and typically

requires domain knowledge in crime investigation/smart traffic planning, we position Clique to solve recall-oriented tasks, i.e., requiring all true cell to be retrieved, and adopt the metric of recall for evaluation.

5.8 Conclusions

We built Clique, a practical object ReID engine that answers spatiotemporal queries. Clique is built upon two unconventional techniques. First, Clique approximates distinct objects by clustering unreliable object features emitted by ReID algorithms before matching with the input image. Second, to search in colossal video data, Clique samples cameras to maximize the spatiotemporal coverage and incrementally adds additional cameras on demand. On 25 hours of city videos spanning 25 cameras, Clique on average reached an accuracy of 0.87 and runs at $830\times$ video real time in achieving high accuracy.

6. CONCLUSIONS AND FUTURE DIRECTIONS

6.1 Conclusions

To support cost-efficient video analytics given colossal video footage generated every day, three software systems are proposed in this thesis. First, we propose VStore that automatically configures video format knobs for retrospective video analytics. VStore explores backward derivation of configuration: it passes the certain video formats desired by analytics backward to retrieval, to storage, and to ingestion. It runs queries at up to $362\times$ of video realtime. Second, we propose DIVA, an analytics engine for querying cold videos on remote, low-cost cameras. At capture time, DIVA builds sparse but accurate landmarks; at query time, it refines query results by continuously updating on-camera vision operators. DIVA effectively answers three types of queries at more than $100\times$ video realtime under typical wireless network and camera hardware. Third, we propose Clique, a practical object ReID engine that answers spatiotemporal object re-identification queries from city-scale cameras. Before matching with the input image, Clique approximates distinct objects by clustering fuzzy object features emitted by ReID algorithms. To search in colossal video data, Clique samples cameras from different locations to maximize the spatiotemporal coverage and incrementally searches in additional cameras on demand. Clique on average reached an accuracy of 0.87 and runs at $830\times$ video real time on 25 hours of city videos spanning 25 cameras.

6.2 Future directions

Supporting more diverse video analytics tasks VStore is one step away from extending itself to support more diverse video analytics tasks. Possible extension could include: (1) supporting more model cascades like vehicle ReID pipelines [46], [181] that typically involve vehicle metadata (e.g., color, make, etc.) classifications before general feature comparisons; (2) supporting object detection/recognition in volumetric videos that detects/classifies objects from 3D video space, which may raise new challenges by adding new types of knobs (e.g., depth information) to the existing fidelity space.

Reducing the scale of camera deployments Deploying multiple cameras per intersection consumes huge human labor, requires high maintenance cost, and raises high privacy concerns. A direction worth exploring is to find a solution that reduces the number of cameras while in the meantime does not sacrifice any query accuracy. Pan-tilt-zoom (PTZ) cameras might be a perfect match. State-of-the-art PTZ cameras have a rotation speed of up to 360° per second, which can effectively capture vehicles from different view angles without losing track of vehicles. In this case, rather than exploring multiple cameras per location incrementally to find vehicle instances with a similar viewpoint with the input image, Clique could directly compare vehicle instances captured from various view angles from a single camera.

REFERENCES

- [1] IHS, *Top video surveillance trends for 2018*, 2018.
- [2] Seagate, *Video surveillance trends report*, <https://www.seagate.com/files/www-content/solutions-content/surveillance-security-video-analytics/en-us/docs/video-surveillance-trends-report.pdf>, 2017.
- [3] *International trends in video surveillance*, <https://cms.uitp.org/wp/wp-content/uploads/2020/06/18-07Statistics-Brief-Videosurveillance-web.pdf>, 2018.
- [4] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. DOI: [10.1038/nature14539](https://doi.org/10.1038/nature14539). [Online]. Available: <https://doi.org/10.1038/nature14539>.
- [5] H. Shen, S. Han, M. Philipose, and A. Krishnamurthy, “Fast video classification via adaptive cascading of deep models,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017.
- [6] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, “Chameleon: Scalable adaptation of video analytics,” in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM ’18, Budapest, Hungary: ACM, 2018, pp. 253–266, ISBN: 978-1-4503-5567-4. DOI: [10.1145/3230543.3230574](https://doi.org/10.1145/3230543.3230574). [Online]. Available: <http://doi.acm.org/10.1145/3230543.3230574>.
- [7] D. Kang, P. Bailis, and M. Zaharia, “Blazeit: Optimizing declarative aggregation and limit queries for neural network-based video analytics,” *Proc. VLDB Endow.*, vol. 13, no. 4, pp. 533–546, Dec. 2019, ISSN: 2150-8097. DOI: [10.14778/3372716.3372725](https://doi.org/10.14778/3372716.3372725). [Online]. Available: <https://doi.org/10.14778/3372716.3372725>.
- [8] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, “Noscope: Optimizing neural network queries over video at scale,” *Proc. VLDB Endow.*, vol. 10, no. 11, pp. 1586–1597, Aug. 2017, ISSN: 2150-8097. DOI: [10.14778/3137628.3137664](https://doi.org/10.14778/3137628.3137664). [Online]. Available: <https://doi.org/10.14778/3137628.3137664>.
- [9] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, “Live video analytics at scale with approximation and delay-tolerance,” in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, Boston, MA: USENIX Association, 2017, pp. 377–392, ISBN: 978-1-931971-37-9. [Online]. Available: <https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zhang>.
- [10] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *arXiv preprint arXiv:1804.02767*, 2018.

- [11] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. Netravali, “Reducto: On-camera filtering for resource-efficient real-time video analytics,” in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM ’20, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 359–376, ISBN: 9781450379557. DOI: [10.1145/3387514.3405874](https://doi.org/10.1145/3387514.3405874). [Online]. Available: <https://doi.org/10.1145/3387514.3405874>.
- [12] C. Canel, T. Kim, G. Zhou, C. Li, H. Lim, D. G. Andersen, M. Kaminsky, and S. R. Dulloor, “Scaling video analytics on constrained edge nodes,” in *Proceedings of the 2nd SysML Conference*, Palo Alto, California, USA, 2019.
- [13] T. Zhang, A. Chowdhery, P. (Bahl, K. Jamieson, and S. Banerjee, “The design and implementation of a wireless video surveillance system,” in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, ser. MobiCom ’15, Paris, France: ACM, 2015, pp. 426–438, ISBN: 978-1-4503-3619-2. DOI: [10.1145/2789168.2790123](https://doi.org/10.1145/2789168.2790123). [Online]. Available: <http://doi.acm.org/10.1145/2789168.2790123>.
- [14] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, “Glimpse: Continuous, real-time object recognition on mobile devices,” in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys ’15, Seoul, South Korea: Association for Computing Machinery, 2015, pp. 155–168, ISBN: 9781450336314. DOI: [10.1145/2809695.2809711](https://doi.org/10.1145/2809695.2809711). [Online]. Available: <https://doi.org/10.1145/2809695.2809711>.
- [15] K. Hsieh, G. Ananthanarayanan, P. Bodik, S. Venkataraman, P. Bahl, M. Philipose, P. B. Gibbons, and O. Mutlu, “Focus: Querying large video datasets with low latency and low cost,” in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, Carlsbad, CA: USENIX Association, 2018. [Online]. Available: <https://www.usenix.org/conference/osdi18/presentation/hsieh>.
- [16] Z. Feng, J. Wang, J. Harkes, P. Pillai, and M. Satyanarayanan, “Eva: An efficient system for exploratory video analysis,” *SysML*, 2018.
- [17] IHS, *Top video surveillance trends for 2016*, 2016.
- [18] T. Xu, L. M. Botelho, and F. X. Lin, “Vstore: A data store for analytics on large videos,” in *Proceedings of the Fourteenth EuroSys Conference 2019*, ser. EuroSys ’19, Dresden, Germany: ACM, 2019, 16:1–16:17, ISBN: 978-1-4503-6281-8. DOI: [10.1145/3302424.3303971](https://doi.org/10.1145/3302424.3303971). [Online]. Available: <http://doi.acm.org/10.1145/3302424.3303971>.
- [19] M. Xu, T. Xu, Y. Liu, X. Liu, G. Huang, and F. X. Lin, “Supporting video queries on zero-streaming cameras,” *CoRR*, vol. abs/1904.12342, 2019. arXiv: [1904.12342](https://arxiv.org/abs/1904.12342). [Online]. Available: <http://arxiv.org/abs/1904.12342>.

- [20] T. Xu, K. Shen, Y. Fu, H. Shi, and F. X. Lin, *Clique: Spatiotemporal object re-identification at the city scale*, 2020. arXiv: 2012.09329 [cs.DB].
- [21] S. Kang, S. Hong, and Y. Won, “Storage technique for real-time streaming of layered video,” *Multimedia Systems*, vol. 15, no. 2, pp. 63–81, Apr. 2009, ISSN: 1432-1882. DOI: 10.1007/s00530-008-0147-8. [Online]. Available: <https://doi.org/10.1007/s00530-008-0147-8>.
- [22] *Wyze camera v2 1080p*, <https://www.wyze.com/product/wyze-cam-v2/>, 2019.
- [23] *Yi home camera*, <https://www.amazon.com/YI-Security-Surveillance-Monitor-Android/dp/B01CW4AR9K>, 2019.
- [24] *Zosi camera*, <https://www.amazon.com/ZOSI-1280TVL-Security-Weatherproof-Surveillance/dp/B01DF6LJZK>, 2019.
- [25] *The state of wifi vs mobile network experience as 5g arrives*, https://www.opensignal.com/sites/opensignal-com/files/data/reports/global/data-2018-11/state_of_wifi_vs_mobile_opensignal_201811.pdf, 2018.
- [26] B. Han, F. Qian, L. Ji, and V. Gopalakrishnan, “Mp-dash: Adaptive video streaming over preference-aware multipath,” in *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT ’16, Irvine, California, USA: ACM, 2016, pp. 129–143, ISBN: 978-1-4503-4292-6. DOI: 10.1145/2999572.2999606. [Online]. Available: <http://doi.acm.org/10.1145/2999572.2999606>.
- [27] *Comcast business internet data plan*, <https://www.business.org/services/internet/comcast-business-internet-review/>, 2019.
- [28] *Tufts: Video security university policy*, <https://publicsafety.tufts.edu/policies/video-security/>, 2014.
- [29] *Video surveillance laws: Video retention requirements by state*, <https://www.verkada.com/blog/surveillance-laws-video-retention-requirements/>, 2018.
- [30] *New case law on retention periods for video surveillance at the workplace*, <https://www.twobirds.com/en/news/articles/2018/germany/new-case-law-on-retention-periods-for-video-surveillance-at-the-workplace>, 2018.
- [31] *The european data protection supervisor video-surveillance guidelines*, https://edps.europa.eu/sites/edp/files/publication/10-03-17_video-surveillance_guidelines_en.pdf, 2010.

- [32] J. M. Hellerstein, P. J. Haas, and H. J. Wang, "Online aggregation," in *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '97, Tucson, Arizona, USA: ACM, 1997, pp. 171–182, ISBN: 0-89791-911-4. DOI: [10.1145/253260.253291](https://doi.org/10.1145/253260.253291). [Online]. Available: <http://doi.acm.org/10.1145/253260.253291>.
- [33] Z. Feng, S. George, J. Harkes, P. Pillai, R. Klatzky, and M. Satyanarayanan, "Eureka: Edge-based discovery of training data for machine learning," *IEEE Internet Computing*, vol. PP, pp. 1–1, Jan. 2019. DOI: [10.1109/MIC.2019.2892941](https://doi.org/10.1109/MIC.2019.2892941).
- [34] A. Augustin, J. Yi, T. Clausen, and W. Townsley, "A study of lora: Long range & low power networks for the internet of things," *Sensors*, vol. 16, no. 9, p. 1466, 2016.
- [35] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, Jun. 2017, pp. 2573–2574. DOI: [10.1109/ICDCS.2017.182](https://doi.org/10.1109/ICDCS.2017.182).
- [36] Y. Wang, W. Wang, J. Zhang, J. Jiang, and K. Chen, "Bridging the edge-cloud barrier for real-time advanced vision analytics," in *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019.
- [37] Z. Tang, M. Naphade, M.-Y. Liu, X. Yang, S. Birchfield, S. Wang, R. Kumar, D. Anastasiu, and J.-N. Hwang, "Cityflow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019.
- [38] Microsoft, *Video analytics towards vision zero*, 2019.
- [39] X. Li and Z. Zhou, "Object re-identification based on deep learning," in. Jun. 2019, ISBN: 978-1-78985-157-1. DOI: [10.5772/intechopen.86564](https://doi.org/10.5772/intechopen.86564).
- [40] M. Naphade, R. Chellappa, D. Anastasiu, A. Sharma, M.-C. Chang, X. Yang, S. Wang, Z. Tang, and L. Zheng, *Ai city challenge*, 2020. [Online]. Available: <https://www.aicitychallenge.org/>.
- [41] L. Zheng, Y. Yang, and A. G. Hauptmann, "Person re-identification: Past, present and future," *CoRR*, vol. abs/1610.02984, 2016. arXiv: [1610.02984](https://arxiv.org/abs/1610.02984). [Online]. Available: <http://arxiv.org/abs/1610.02984>.
- [42] Y. Sun, L. Zheng, Y. Yang, Q. Tian, and S. Wang, "Beyond part models: Person retrieval with refined part pooling (and a strong convolutional baseline)," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Sep. 2018.

- [43] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian, “Scalable person re-identification: A benchmark,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1116–1124. DOI: [10.1109/ICCV.2015.133](https://doi.org/10.1109/ICCV.2015.133).
- [44] Z. Zhong, L. Zheng, D. Cao, and S. Li, “Re-ranking person re-identification with k-reciprocal encoding,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 3652–3661. DOI: [10.1109/CVPR.2017.389](https://doi.org/10.1109/CVPR.2017.389).
- [45] Y. Fu, Y. Wei, Y. Zhou, H. Shi, G. Huang, X. Wang, Z. Yao, and T. Huang, “Horizontal pyramid matching for person re-identification,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 8295–8302, Jul. 2019. DOI: [10.1609/aaai.v33i01.33018295](https://doi.org/10.1609/aaai.v33i01.33018295). [Online]. Available: <https://ojs.aaai.org/index.php/AAAI/article/view/4842>.
- [46] X. Tan, Z. Wang, M. Jiang, X. Yang, J. Wang, Y. Gao, X. Su, X. Ye, Y. Yuan, D. He, S. Wen, and E. Ding, “Multi-camera vehicle tracking and re-identification based on visual and spatial-temporal features,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2019.
- [47] T.-W. Huang, J. Cai, H. Yang, H.-M. Hsu, and J.-N. Hwang, “Multi-view vehicle re-identification using temporal attention model and metadata re-ranking,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2019.
- [48] Y. Fu, Y. Wei, G. Wang, Y. Zhou, H. Shi, and T. S. Huang, “Self-similarity grouping: A simple unsupervised cross domain adaptation approach for person re-identification,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2019.
- [49] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.
- [50] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 779–788. DOI: [10.1109/CVPR.2016.91](https://doi.org/10.1109/CVPR.2016.91).
- [51] W. MathWorld, L^2 norm, <https://mathworld.wolfram.com/L2-Norm.html>, 2020.
- [52] K. Lv, H. Du, Y. Hou, W. Deng, H. Sheng, J. Jiao, and L. Zheng, “Vehicle re-identification with location and time stamps,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2019.

- [53] C. for the New Urbanism, *Street networks 101*, <https://www.cnu.org/our-projects/street-networks/street-networks-101>, 2020.
- [54] J. Redmon and A. Farhadi, “Yolo9000: Better, faster, stronger,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6517–6525. DOI: [10.1109/CVPR.2017.690](https://doi.org/10.1109/CVPR.2017.690).
- [55] J. Redmon and A. Farhadi, *Yolo: Real-time object detection*, 2016. [Online]. Available: <https://pjreddie.com/darknet/yolov2/>.
- [56] S. Jain, J. Jiang, Y. Shu, G. Ananthanarayanan, and J. Gonzalez, “Rexcam: Resource-efficient, cross-camera video analytics at enterprise scale,” *CoRR*, vol. abs/1811.01268, 2018. arXiv: [1811.01268](https://arxiv.org/abs/1811.01268). [Online]. Available: <http://arxiv.org/abs/1811.01268>.
- [57] S. Jain, G. Ananthanarayanan, J. Jiang, Y. Shu, and J. E. Gonzalez, “Scaling video analytics systems to large camera deployments,” *arXiv preprint arXiv:1809.02318*, 2018.
- [58] B. Audeh, P. Beaune, and M. Beigbeder, “Recall-oriented evaluation for information retrieval systems,” in *Multidisciplinary Information Retrieval*, M. Lupu, E. Kanoulas, and F. Loizides, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 29–32, ISBN: 978-3-642-41057-4.
- [59] A. Arampatzis, J. Kamps, and S. Robertson, “Where to stop reading a ranked list? threshold optimization using truncated score distributions,” in *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR ’09, Boston, MA, USA: Association for Computing Machinery, 2009, pp. 524–531, ISBN: 9781605584836. DOI: [10.1145/1571941.1572031](https://doi.org/10.1145/1571941.1572031). [Online]. Available: <https://doi.org/10.1145/1571941.1572031>.
- [60] D. Bahri, Y. Tay, C. Zheng, D. Metzler, and A. Tomkins, “Choppy: Cut transformer for ranked list truncation,” in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR ’20, Virtual Event, China: Association for Computing Machinery, 2020, pp. 1513–1516, ISBN: 9781450380164. DOI: [10.1145/3397271.3401188](https://doi.org/10.1145/3397271.3401188). [Online]. Available: <https://doi.org/10.1145/3397271.3401188>.
- [61] X. Jin and J. Han, “K-means clustering,” in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 563–564, ISBN: 978-0-387-30164-8. DOI: [10.1007/978-0-387-30164-8_425](https://doi.org/10.1007/978-0-387-30164-8_425). [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_425.

- [62] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, “An efficient k-means clustering algorithm: Analysis and implementation,” vol. 24, no. 7, 2002, ISSN: 0162-8828. DOI: [10.1109/TPAMI.2002.1017616](https://doi.org/10.1109/TPAMI.2002.1017616). [Online]. Available: <https://doi.org/10.1109/TPAMI.2002.1017616>.
- [63] S. C. Johnson, “Hierarchical clustering schemes,” *Psychometrika*, vol. 32, pp. 241–254, 1967.
- [64] OpenALPR Technology, Inc., *Openalpr*, <https://github.com/openalpr/openalpr>, 2018.
- [65] N. Agrawal and A. Vulimiri, “Low-latency analytics on colossal data streams with summarystore,” in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP ’17, Shanghai, China: ACM, 2017, pp. 647–664, ISBN: 978-1-4503-5085-3. DOI: [10.1145/3132747.3132758](https://doi.org/10.1145/3132747.3132758). [Online]. Available: <http://doi.acm.org/10.1145/3132747.3132758>.
- [66] . Oracle, *Dramatically reduce the cost and complexity of video surveillance storage*, <https://www.oracle.com/assets/wp-video-surveillance-storage-2288409.pdf>, 2015.
- [67] S. Modiri Assari, H. Idrees, and M. Shah, “Human re-identification in crowd videos using personal, social and environmental constraints,” in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Cham: Springer International Publishing, 2016, pp. 119–136, ISBN: 978-3-319-46475-6.
- [68] Y. Chen, X. Zhu, W. Zheng, and J. Lai, “Person re-identification by camera correlation aware feature augmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 2, pp. 392–408, Feb. 2018, ISSN: 0162-8828. DOI: [10.1109/TPAMI.2017.2666805](https://doi.org/10.1109/TPAMI.2017.2666805).
- [69] Y. Wu, J. Lim, and M.-H. Yang, “Online object tracking: A benchmark,” in *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR ’13, Washington, DC, USA: IEEE Computer Society, 2013, pp. 2411–2418, ISBN: 978-0-7695-4989-7. DOI: [10.1109/CVPR.2013.312](https://doi.org/10.1109/CVPR.2013.312). [Online]. Available: <http://dx.doi.org/10.1109/CVPR.2013.312>.
- [70] R. Girshick, “Fast r-cnn,” in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ser. ICCV ’15, Washington, DC, USA: IEEE Computer Society, 2015, pp. 1440–1448, ISBN: 978-1-4673-8391-2. DOI: [10.1109/ICCV.2015.169](https://doi.org/10.1109/ICCV.2015.169). [Online]. Available: <http://dx.doi.org/10.1109/ICCV.2015.169>.

- [71] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS’15, Montreal, Canada: MIT Press, 2015, pp. 91–99. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2969239.2969250>.
- [72] C.-Y. Wu, M. Zaheer, H. Hu, R. Manmatha, A. J. Smola, and P. Krähenbühl, “Compressed video action recognition,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2018.
- [73] S. Fouladi, R. S. Wahby, B. Shacklett, K. V. Balasubramaniam, W. Zeng, R. Bhalerao, A. Sivaraman, G. Porter, and K. Winstein, “Encoding, fast and slow: Low-latency video processing using thousands of tiny threads,” in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, Boston, MA: USENIX Association, 2017, pp. 363–376, ISBN: 978-1-931971-37-9. [Online]. Available: <https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi>.
- [74] *Rollingdb storage library*, <https://github.com/openalpr/rollingdb>, 2018.
- [75] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hossfeld, and P. Tran-Gia, “A survey on quality of experience of http adaptive streaming,” *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 469–492, 2015.
- [76] C. Kreuzberger, D. Posch, and H. Hellwagner, “A scalable video coding dataset and toolchain for dynamic adaptive streaming over http,” in *Proceedings of the 6th ACM Multimedia Systems Conference*, ser. MMSys ’15, Portland, Oregon: ACM, 2015, pp. 213–218, ISBN: 978-1-4503-3351-1. DOI: [10.1145/2713168.2713193](https://doi.org/10.1145/2713168.2713193). [Online]. Available: <http://doi.acm.org/10.1145/2713168.2713193>.
- [77] OpenCV, *Optical flow*, 2018.
- [78] OpenCV, *Contours*, 2018.
- [79] Y. Cheng, X. Sun, and Y. L. Yin, “Searching monotone multi-dimensional arrays,” *Discrete Mathematics*, vol. 308, no. 11, pp. 2213–2221, 2008.
- [80] N. Linial and M. Saks, “Searching ordered structures,” *Journal of algorithms*, vol. 6, no. 1, pp. 86–103, 1985.
- [81] E. T. Bell, “Exponential polynomials,” *Annals of Mathematics*, vol. 35, no. 2, pp. 258–277, 1934, ISSN: 0003486X. [Online]. Available: <http://www.jstor.org/stable/1968431>.

- [82] E. T. Bell, “Exponential numbers,” *The American Mathematical Monthly*, vol. 41, no. 7, pp. 411–419, 1934, ISSN: 00029890, 19300972. [Online]. Available: <http://www.jstor.org/stable/2300300>.
- [83] D. Bertsekas and R. Gallager, *Data Networks (2Nd Ed.)* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1992, ISBN: 0-13-200916-1.
- [84] I. Molnár, *[patch] modular scheduler core and completely fair scheduler*, <http://lwn.net/Articles/230501/>, 2007.
- [85] iMatix Corporation, *Lightning memory-mapped database*, <https://symas.com/lmdb/>, 2018.
- [86] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, “Tensorflow: A system for large-scale machine learning,” in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Savannah, GA: USENIX Association, 2016, pp. 265–283, ISBN: 978-1-931971-33-1. [Online]. Available: <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi>.
- [87] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.
- [88] P. Liu, J. Yoon, L. Johnson, and S. Banerjee, “Greening the video transcoding service with low-cost hardware transcoders,” in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, Denver, CO: USENIX Association, 2016, pp. 407–419, ISBN: 978-1-931971-30-0. [Online]. Available: <https://www.usenix.org/conference/atc16/technical-sessions/presentation/liu>.
- [89] J. Yoon, P. Liu, and S. Banerjee, “Low-cost video transcoding at the wireless edge,” in *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, Oct. 2016, pp. 129–141. DOI: [10.1109/SEC.2016.8](https://doi.org/10.1109/SEC.2016.8).
- [90] *Amazon ec2 p3 instances*, <https://aws.amazon.com/ec2/instance-types/p3/>, 2018.

- [91] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, “Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints,” in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys ’16, Singapore, Singapore: ACM, 2016, pp. 123–136, ISBN: 978-1-4503-4269-8. DOI: [10.1145/2906388.2906396](https://doi.org/10.1145/2906388.2906396). [Online]. Available: <http://doi.acm.org/10.1145/2906388.2906396>.
- [92] C.-C. Hung, G. Ananthanarayanan, P. Bodík, L. Golubchik, M. Yu, V. Bahl, and M. Philipose, “Videoedge: Processing camera streams using hierarchical clusters,” Oct. 2018. [Online]. Available: <https://www.microsoft.com/en-us/research/publication/videoedge-processing-camera-streams-using-hierarchical-clusters/>.
- [93] C. Pakha, A. Chowdhery, and J. Jiang, “Reinventing video streaming for distributed vision analytics,” in *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*, Boston, MA: USENIX Association, 2018. [Online]. Available: <https://www.usenix.org/conference/hotcloud18/presentation/pakha>.
- [94] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel, “Finding a needle in haystack: Facebook’s photo storage,” in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI’10, Vancouver, BC, Canada: USENIX Association, 2010, pp. 47–60. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1924943.1924947>.
- [95] V. Gupta-Cledat, L. Remis, and C. R. Strong, “Addressing the dark side of vision research: Storage,” in *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17)*, Santa Clara, CA: USENIX Association, 2017. [Online]. Available: <https://www.usenix.org/conference/hotstorage17/program/presentation/gupta-cledat>.
- [96] L. Remis, V. Gupta-Cledat, C. R. Strong, and M. IJzerman-Korevaar, “Vdms: An efficient big-visual-data access for machine learning workloads,” *CoRR*, vol. abs/1810.11832, 2018.
- [97] S. Papadopoulos, K. Datta, S. Madden, and T. Mattson, “The tiledb array data storage manager,” *Proc. VLDB Endow.*, vol. 10, no. 4, pp. 349–360, Nov. 2016, ISSN: 2150-8097. DOI: [10.14778/3025111.3025117](https://doi.org/10.14778/3025111.3025117). [Online]. Available: <https://doi.org/10.14778/3025111.3025117>.
- [98] J. Casper, J. Barker, and B. Catanzaro, *Nvvl: Nvidia video loader*, <https://github.com/NVIDIA/nvvl>, 2018.

- [99] A. Poms, W. Crichton, P. Hanrahan, and K. Fatahalian, “Scanner: Efficient video analysis at scale,” *ACM Trans. Graph.*, vol. 37, no. 4, 138:1–138:13, Jul. 2018, ISSN: 0730-0301. DOI: [10.1145/3197517.3201394](https://doi.acm.org/10.1145/3197517.3201394). [Online]. Available: <http://doi.acm.org/10.1145/3197517.3201394>.
- [100] *Nvidia*, <https://developer.nvidia.com/deepstream-sdk>, 2018.
- [101] M. P. Andersen and D. E. Culler, “Btrdb: Optimizing storage system design for timeseries processing,” in *14th USENIX Conference on File and Storage Technologies (FAST 16)*, Santa Clara, CA: USENIX Association, 2016, pp. 39–52, ISBN: 978-1-931971-28-7. [Online]. Available: <https://www.usenix.org/conference/fast16/technical-sessions/presentation/andersen>.
- [102] V. Srinivasan, B. Bulkowski, W.-L. Chu, S. Sayyaparaju, A. Gooding, R. Iyer, A. Shinde, and T. Lopatic, “Aerospike: Architecture of a real-time operational dbms,” *Proc. VLDB Endow.*, vol. 9, no. 13, pp. 1389–1400, Sep. 2016, ISSN: 2150-8097. DOI: [10.14778/3007263.3007276](https://dx.doi.org/10.14778/3007263.3007276). [Online]. Available: <http://dx.doi.org/10.14778/3007263.3007276>.
- [103] InfluxData, *Influxdb*, <https://www.influxdata.com/>, 2018.
- [104] T. K. Sellis, “Multiple-query optimization,” *ACM Trans. Database Syst.*, vol. 13, no. 1, pp. 23–52, Mar. 1988, ISSN: 0362-5915. DOI: [10.1145/42201.42203](https://doi.acm.org/10.1145/42201.42203). [Online]. Available: <http://doi.acm.org/10.1145/42201.42203>.
- [105] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, “Models and issues in data stream systems,” in *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS ’02, Madison, Wisconsin: ACM, 2002, pp. 1–16, ISBN: 1-58113-507-6. DOI: [10.1145/543613.543615](https://doi.acm.org/10.1145/543613.543615). [Online]. Available: <http://doi.acm.org/10.1145/543613.543615>.
- [106] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma, “Query processing, resource management, and approximation in a data stream management system,” in *IN CIDR*, 2003, pp. 245–256.
- [107] L. Qiao, V. Raman, F. Reiss, P. J. Haas, and G. M. Lohman, “Main-memory scan sharing for multi-core cpus,” *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 610–621, Aug. 2008, ISSN: 2150-8097. DOI: [10.14778/1453856.1453924](https://dx.doi.org/10.14778/1453856.1453924). [Online]. Available: <http://dx.doi.org/10.14778/1453856.1453924>.

- [108] M. Zukowski, S. Héman, N. Nes, and P. Boncz, “Cooperative scans: Dynamic bandwidth sharing in a dbms,” in *Proceedings of the 33rd International Conference on Very Large Data Bases*, ser. VLDB '07, Vienna, Austria: VLDB Endowment, 2007, pp. 723–734, ISBN: 978-1-59593-649-3. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1325851.1325934>.
- [109] C. A. Lang, B. Bhattacharjee, T. Malkemus, S. Padmanabhan, and K. Wong, “Increasing buffer-locality for multiple relational table scans through grouping and throttling,” in *2007 IEEE 23rd International Conference on Data Engineering*, Apr. 2007, pp. 1136–1145. DOI: [10.1109/ICDE.2007.368972](https://doi.org/10.1109/ICDE.2007.368972).
- [110] N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker, “Load shedding in a data stream manager,” in *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, ser. VLDB '03, Berlin, Germany: VLDB Endowment, 2003, pp. 309–320, ISBN: 0-12-722442-4. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1315451.1315479>.
- [111] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik, “Monitoring streams: A new class of data management applications,” in *Proceedings of the 28th International Conference on Very Large Data Bases*, ser. VLDB '02, Hong Kong, China: VLDB Endowment, 2002, pp. 215–226. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1287369.1287389>.
- [112] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, “Aurora: A new model and architecture for data stream management,” *The VLDB Journal*, vol. 12, no. 2, pp. 120–139, Aug. 2003, ISSN: 1066-8888. DOI: [10.1007/s00778-003-0095-z](https://doi.org/10.1007/s00778-003-0095-z). [Online]. Available: <http://dx.doi.org/10.1007/s00778-003-0095-z>.
- [113] K. Keeton and R. H. Katz, “Evaluating video layout strategies for a high-performance storage server,” *Multimedia Systems*, vol. 3, no. 2, pp. 43–52, May 1995, ISSN: 1432-1882. DOI: [10.1007/BF01219800](https://doi.org/10.1007/BF01219800). [Online]. Available: <https://doi.org/10.1007/BF01219800>.
- [114] T.-c. Chiueh and R. H. Katz, “Multi-resolution video representation for parallel disk arrays,” in *Proceedings of the First ACM International Conference on Multimedia*, ser. MULTIMEDIA '93, Anaheim, California, USA: ACM, 1993, pp. 401–409, ISBN: 0-89791-596-8. DOI: [10.1145/166266.168438](https://doi.org/10.1145/166266.168438). [Online]. Available: <http://doi.acm.org/10.1145/166266.168438>.

- [115] J. Oh and K. A. Hua, “Efficient and cost-effective techniques for browsing and indexing large video databases,” in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD ’00, Dallas, Texas, USA: ACM, 2000, pp. 415–426, ISBN: 1-58113-217-4. DOI: [10.1145/342009.335436](https://doi.org/10.1145/342009.335436). [Online]. Available: <http://doi.acm.org/10.1145/342009.335436>.
- [116] Q. Huang, P. Ang, P. Knowles, T. Nykiel, I. Tverdokhlib, A. Yajurvedi, P. Dapolito IV, X. Yan, M. Bykov, C. Liang, M. Talwar, A. Mathur, S. Kulkarni, M. Burke, and W. Lloyd, “Sve: Distributed video processing at facebook scale,” in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP ’17, Shanghai, China: ACM, 2017, pp. 87–103, ISBN: 978-1-4503-5085-3. DOI: [10.1145/3132747.3132775](https://doi.org/10.1145/3132747.3132775). [Online]. Available: <http://doi.acm.org/10.1145/3132747.3132775>.
- [117] H. Wang, K. Rudy, J. Li, and D. Ni, “Calculation of traffic flow breakdown probability to optimize link throughput,” *Applied Mathematical Modelling*, vol. 34, no. 11, pp. 3376–3389, 2010, ISSN: 0307-904X. DOI: <https://doi.org/10.1016/j.apm.2010.02.027>. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0307904X10000880>.
- [118] S. William Saletan, *The case for mass surveillance*, https://www.delcotimes.com/news/the-case-for-mass-surveillance/article__61a27a3c-8e8b-54e3-b048-e44682b6a024.html, 2013.
- [119] H. Law and J. Deng, “Cornernet: Detecting objects as paired keypoints,” *International Journal of Computer Vision (IJCV)*, Aug. 2019, ISSN: 1573-1405. DOI: [10.1007/s11263-019-01204-1](https://doi.org/10.1007/s11263-019-01204-1). [Online]. Available: <https://doi.org/10.1007/s11263-019-01204-1>.
- [120] *Wyze camera specifications*, <https://www.wyze.com/wyze-cam/specs/>, 2019.
- [121] *Hisilicon ip camera specifications*. <http://www.hisilicon.com/en/Products/ProductList/Surveillance>, 2019.
- [122] M. Liao, *Benchmarking hardware for cnn inference in 2018*, <https://towardsdatascience.com/benchmarking-hardware-for-cnn-inference-in-2018-1d58268de12a>, 2018.
- [123] *Running yolo detection on raspberry pi*. <http://raspberrypi4u.blogspot.com/2018/10/raspberry-pi-yolo-real-time-object.html>, 2018.
- [124] *Wireless cameras slowing router too much*, <https://community.netgear.com/t5/Nighthawk-WiFi-Routers/Wireless-cameras-slowing-router-too-much/td-p/513047>, 2015.

- [125] *Wifi cameras*, <https://www.security-camera-warehouse.com/ip-camera/wifi-enabled/>, 2018.
- [126] *The zettabyte era: Trends and analysis*, <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html>, 2017.
- [127] NIST, *The spectrum crunch*, 2019. [Online]. Available: <https://www.nist.gov/topics/advanced-communications/spectrum-crunch>.
- [128] *Understanding ip surveillance camera bandwidth*, <https://www.fortinet.com/content/dam/fortinet/assets/white-papers/wp-ip-surveillance-camera.pdf>, 2017.
- [129] K. Chakrabarti, K. Porkaew, and S. Mehrotra, “Efficient query refinement in multimedia databases,” in *ICDE Conference*, Poster paper, Jan. 2000. [Online]. Available: <https://www.microsoft.com/en-us/research/publication/efficient-query-refinement-in-multimedia-databases/>.
- [130] I. F. Ilyas, R. Shah, W. G. Aref, J. S. Vitter, and A. K. Elmagarmid, “Rank-aware query optimization,” in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data (ICMD)*, 2004, pp. 203–214.
- [131] C. Böhm, B. Braunmüller, F. Krebs, and H.-P. Kriegel, “Epsilon grid order: An algorithm for the similarity join on massive high-dimensional data,” in *ACM SIGMOD Record*, vol. 30, 2001, pp. 379–388.
- [132] N. Koudas and K. C. Sevcik, “High dimensional similarity joins: Algorithms and performance evaluation,” *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 12, no. 1, pp. 3–18, 2000.
- [133] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, “Mapreduce online,” in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI’10, San Jose, California: USENIX Association, 2010, pp. 21–21. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1855711.1855732>.
- [134] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie, “Online aggregation for large mapreduce jobs,” *Proceedings of the VLDB Endowment*, vol. 4, no. 11, pp. 1135–1145, 2011.
- [135] V. Saligrama and Z. Chen, “Video anomaly detection based on local statistical aggregates,” *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2112–2119, 2012.

- [136] X. Zhu, J. Dai, L. Yuan, and Y. Wei, “Towards high performance video object detection,” in *CVPR*, IEEE Computer Society, 2018, pp. 7210–7218.
- [137] T. Blu, P. Dragotti, M. Vetterli, P. Marziliano, and L. Coulot, “Sparse sampling of signal innovations,” *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 31–40, Mar. 2008, ISSN: 1053-5888. DOI: [10.1109/MSP.2007.914998](https://doi.org/10.1109/MSP.2007.914998).
- [138] C. R. Shalizi, *Advanced data analysis from an elementary point of view*, <http://www.stat.cmu.edu/~cshalizi/ADAfaEPoV/ADAfaEPoV.pdf>, 2019.
- [139] *Background subtraction*, https://docs.opencv.org/3.4.0/db/d5c/tutorial_py_bg_subtraction.html, 2019.
- [140] *Youtube live streaming: Jackson hole*, <https://youtu.be/2wnU2Kp7quQ>, 2019.
- [141] *Youtube live streaming: Jackson town*, <https://www.youtube.com/watch?v=1EiC9bvVGnk>, 2019.
- [142] *Youtube live streaming: Banff*, <https://youtu.be/9HwSNGcdQ7k>, 2019.
- [143] *Youtube live streaming: Mierlo*, <https://www.youtube.com/watch?v=HbtBgxFkDHU>, 2019.
- [144] *Youtube live streaming: Miami*, <https://www.youtube.com/watch?v=0dctq-YjAdc>, 2019.
- [145] *Youtube live streaming: Ashland*, <https://www.youtube.com/watch?v=e47XhLmZhFk>, 2019.
- [146] *Youtube live streaming: Shibuya*, <https://youtu.be/PmrWwYTlAVQ>, 2019.
- [147] *Youtube live streaming: Chaweng*, https://www.youtube.com/watch?v=tihJ58_qiH0, 2019.
- [148] *Youtube live streaming: Lausanne*, <https://www.youtube.com/watch?v=7uF7DsUQ9vc>, 2019.
- [149] *Youtube live streaming: Venice*, <https://www.youtube.com/watch?v=JqUREqYduHw>, 2019.
- [150] *Youtube live streaming: Oxford*, <https://www.youtube.com/watch?v=St7aTfoIdYQ>, 2019.

- [151] *Youtube live streaming: Whitebay*, <https://www.youtube.com/watch?v=LXWVYoBluT4>, 2019.
- [152] *Youtube live streaming: Coralreef*, <https://youtu.be/WYOe8SfQbac>, 2019.
- [153] *Youtube live streaming: Boathouse*, <https://www.youtube.com/watch?v=TXw7CyY0TbU&t=0s>, 2019.
- [154] *Youtube live streaming: Eagle*, https://www.youtube.com/watch?v=Q__OrM8o2k6I, 2019.
- [155] T. Jin and S. Hong, “Split-cnn: Splitting window-based operations in convolutional neural networks for memory system optimization,” in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019, pp. 835–847.
- [156] *Build intelligent ideas with our platform for local ai*, <https://coral.withgoogle.com/>, 2019.
- [157] E. Teng, J. D. Falcão, and B. Iannucci, “Clickbait: Click-based accelerated incremental training of convolutional neural networks,” *CoRR*, vol. abs/1709.05021, 2017. arXiv: [1709.05021](https://arxiv.org/abs/1709.05021). [Online]. Available: <http://arxiv.org/abs/1709.05021>.
- [158] E. Teng, R. Huang, and B. Iannucci, “Clickbait-v2: Training an object detector in real-time,” *CoRR*, vol. abs/1803.10358, 2018. arXiv: [1803.10358](https://arxiv.org/abs/1803.10358). [Online]. Available: <http://arxiv.org/abs/1803.10358>.
- [159] C. Käding, E. Rodner, A. Freytag, and J. Denzler, “Fine-tuning deep neural networks in continuous learning scenarios,” in *Computer Vision – ACCV 2016 Workshops*, C.-S. Chen, J. Lu, and K.-K. Ma, Eds., Cham: Springer International Publishing, 2017, pp. 588–605, ISBN: 978-3-319-54526-4.
- [160] M. Liu and M. Zhu, “Mobile video object detection with temporally-aware feature maps,” *CVPR*, 2018.
- [161] K. Kang, W. Ouyang, H. Li, and X. Wang, “Object detection from video tubelets with convolutional neural networks,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 817–825. DOI: [10.1109/CVPR.2016.95](https://doi.org/10.1109/CVPR.2016.95).
- [162] B. Feng, K. Wan, S. Yang, and Y. Ding, “SECS: efficient deep stream processing via class skew dichotomy,” *CoRR*, vol. abs/1809.06691, 2018. arXiv: [1809.06691](https://arxiv.org/abs/1809.06691). [Online]. Available: <http://arxiv.org/abs/1809.06691>.

- [163] S. Krishnan, A. Dziedzic, and A. J. Elmore, “Deeplens: Towards a visual data management system,” in *CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*, 2019. [Online]. Available: <http://cidrdb.org/cidr2019/papers/p40-krishnan-cidr19.pdf>.
- [164] H. Shen, L. Chen, Y. Jin, L. Zhao, B. Kong, M. Philipose, A. Krishnamurthy, and R. Sundaram, “Nexus: A gpu cluster engine for accelerating dnn-based video analysis,” in *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, 2019, pp. 322–337.
- [165] P. Viola, M. Jones, *et al.*, “Rapid object detection using a boosted cascade of simple features,” *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition (CVPR)*, vol. 1, pp. 511–518, 2001.
- [166] Z. Cai, M. Saberian, and N. Vasconcelos, “Learning complexity-aware cascades for deep pedestrian detection,” in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 3361–3369.
- [167] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, “Deepdecision: A mobile deep learning framework for edge video analytics,” in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, Apr. 2018, pp. 1421–1429. DOI: [10.1109/INFOCOM.2018.8485905](https://doi.org/10.1109/INFOCOM.2018.8485905).
- [168] J. Wang, Z. Feng, Z. Chen, S. George, M. Bala, P. Pillai, S. Yang, and M. Satyanarayanan, “Bandwidth-efficient live video analytics for drones via edge computing,” in *2018 IEEE/ACM Symposium on Edge Computing, SEC 2018, Seattle, WA, USA, October 25-27, 2018*, 2018, pp. 159–173. DOI: [10.1109/SEC.2018.00019](https://doi.org/10.1109/SEC.2018.00019). [Online]. Available: <https://doi.org/10.1109/SEC.2018.00019>.
- [169] M. Xu, X. Zhang, Y. Liu, G. Huang, X. Liu, and F. X. Lin, “Approximate query service on autonomous iot cameras,” in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020, pp. 191–205.
- [170] J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. J. Haas, “Interactive data analysis: The control project,” *Computer*, vol. 32, no. 8, pp. 51–59, Aug. 1999, ISSN: 0018-9162. DOI: [10.1109/2.781635](https://doi.org/10.1109/2.781635).
- [171] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, K. Karanasos, J. Padhye, and G. Varghese, “Wanalytics: Geo-distributed analytics for a data intensive world,” in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD ’15, Melbourne, Victoria, Australia: ACM, 2015, pp. 1087–1092, ISBN: 978-1-4503-2758-9. DOI: [10.1145/2723372.2735365](https://doi.org/10.1145/2723372.2735365). [Online]. Available: <http://doi.acm.org/10.1145/2723372.2735365>.

- [172] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, “Global analytics in the face of bandwidth and regulatory constraints,” in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, Oakland, CA: USENIX Association, 2015, pp. 323–336, ISBN: 978-1-931971-218. [Online]. Available: <https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/vulimiri>.
- [173] R. Viswanathan, G. Ananthanarayanan, and A. Akella, “CLARINET: Wan-aware optimization for analytics queries,” in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Savannah, GA: USENIX Association, 2016, pp. 435–450, ISBN: 978-1-931971-33-1. [Online]. Available: <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/viswanathan>.
- [174] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, “Low latency geo-distributed data analytics,” *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 421–434, Aug. 2015, ISSN: 0146-4833. DOI: [10.1145/2829988.2787505](https://doi.org/10.1145/2829988.2787505). [Online]. Available: <http://doi.acm.org/10.1145/2829988.2787505>.
- [175] H. Wang and B. Li, “Lube: Mitigating bottlenecks in wide area data analytics,” in *9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 17)*, Santa Clara, CA: USENIX Association, 2017. [Online]. Available: <https://www.usenix.org/conference/hotcloud17/program/presentation/wang>.
- [176] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman, “Aggregation and degradation in jetstream: Streaming analytics in the wide area,” in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, Seattle, WA: USENIX Association, 2014, pp. 275–288, ISBN: 978-1-931971-09-6. [Online]. Available: <https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/rabkin>.
- [177] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, “Awstream: Adaptive wide-area streaming analytics,” in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM ’18, Budapest, Hungary: ACM, 2018, pp. 236–252, ISBN: 978-1-4503-5567-4. DOI: [10.1145/3230543.3230554](https://doi.org/10.1145/3230543.3230554). [Online]. Available: <http://doi.acm.org/10.1145/3230543.3230554>.
- [178] P. with code, *Person re-identification on dukemtmc-reid*, 2020. [Online]. Available: <https://paperswithcode.com/sota/person-re-identification-on-dukemtmc-reid>.
- [179] M. Gou, S. Karanam, W. Liu, O. Camps, and R. J. Radke, “Dukemtmc4reid: A large-scale multi-camera person re-identification dataset,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Jul. 2017, pp. 1425–1434. DOI: [10.1109/CVPRW.2017.185](https://doi.org/10.1109/CVPRW.2017.185).

- [180] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian, “Scalable person re-identification: A benchmark,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1116–1124.
- [181] D. Huang, V. Ramanathan, D. Mahajan, M. Paluri, L. Fei-Fei, and J. C. Niebles, “What makes a video a video: Analyzing temporal information in video understanding models and datasets,” in *CVPR*, IEEE Computer Society, 2018, pp. 7366–7375.
- [182] M. Xu, X. Zhang, Y. Liu, G. Huang, X. Liu, and F. X. Lin, “Approximate query service on autonomous iot cameras,” in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys ’20, Toronto, Ontario, Canada: Association for Computing Machinery, 2020, pp. 191–205, ISBN: 9781450379540. DOI: [10.1145/3386901.3388948](https://doi.org/10.1145/3386901.3388948). [Online]. Available: <https://doi.org/10.1145/3386901.3388948>.
- [183] OpenCV, *Histogram calculation*, docs.opencv.org.
- [184] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.
- [185] L. Cheng and J. Wang, “Vittrack: Efficient tracking on the edge for commodity video surveillance systems,” in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 1052–1060. DOI: [10.1109/INFOCOM.2018.8486353](https://doi.org/10.1109/INFOCOM.2018.8486353).
- [186] J. N. Hughes, A. Annex, C. N. Eichelberger, A. Fox, A. Hulbert, and M. Ronquest, “GeoMesa: a distributed architecture for spatio-temporal fusion,” in *Geospatial Informatics, Fusion, and Motion Video Analytics V*, M. F. Pallechia, K. Palaniappan, P. J. Doucette, S. L. Dockstader, G. Seetharaman, and P. B. Deignan, Eds., International Society for Optics and Photonics, vol. 9473, SPIE, 2015, pp. 128–140. DOI: [10.1117/12.2177233](https://doi.org/10.1117/12.2177233). [Online]. Available: <https://doi.org/10.1117/12.2177233>.
- [187] M. Erwig, R. H. Güting, M. Schneider, and M. Vazirgiannis, “Spatio-temporal data types: An approach to modeling and querying moving objects in databases,” *Geoinformatica*, vol. 3, no. 3, pp. 269–296, Sep. 1999, ISSN: 1384-6175. DOI: [10.1023/A:1009805532638](https://doi.org/10.1023/A:1009805532638). [Online]. Available: <https://doi.org/10.1023/A:1009805532638>.
- [188] T. Abraham and J. Roddick, “Survey of spatio-temporal databases,” *GeoInformatica*, vol. 3, pp. 61–99, 1999.
- [189] N. Pant, M. Fouladgar, R. Elmasri, and K. Jitkajornwanich, “A survey of spatio-temporal database research,” in *Intelligent Information and Database Systems*, N. T. Nguyen, D. H. Hoang, T.-P. Hong, H. Pham, and B. Trawiński, Eds., Cham: Springer International Publishing, 2018, pp. 115–126, ISBN: 978-3-319-75420-8.

- [190] K. Porkaew, I. Lazaridis, and S. Mehrotra, “Querying mobile objects in spatio-temporal databases,” in *Advances in Spatial and Temporal Databases*, C. S. Jensen, M. Schneider, B. Seeger, and V. J. Tsotras, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 59–78, ISBN: 978-3-540-47724-2.
- [191] sklearn, *Kernel ridge regression*, 2020. [Online]. Available: https://scikit-learn.org/stable/modules/kernel_ridge.html.
- [192] Z. Zhu, Q. Wang, B. Li, W. Wu, J. Yan, and W. Hu, “Distractor-aware siamese networks for visual object tracking,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, Sep. 2018.
- [193] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu, “High performance visual tracking with siamese region proposal network,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8971–8980.
- [194] “Deep learning in video multi-object tracking: A survey,” *Neurocomputing*, vol. 381, pp. 61–88, 2020, ISSN: 0925-2312. DOI: <https://doi.org/10.1016/j.neucom.2019.11.023>.
- [195] D. Shah, *The surveillance phenomenon you must know about : Multi object tracking*, <https://medium.com/visionwizard/object-tracking-675d7a33e687>, 2020.
- [196] B. Haynes, A. Mazumdar, M. Balazinska, L. Ceze, and A. Cheung, “Visual road: A video data management benchmark,” in *SIGMOD*, 2019, pp. 972–987. DOI: [10.1145/3299869.3324955](https://doi.org/10.1145/3299869.3324955).
- [197] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, “3d object representations for fine-grained categorization,” ser. ICCVW ’13, USA: IEEE Computer Society, 2013, pp. 554–561, ISBN: 9781479930227. DOI: [10.1109/ICCVW.2013.77](https://doi.org/10.1109/ICCVW.2013.77). [Online]. Available: <https://doi.org/10.1109/ICCVW.2013.77>.
- [198] X. Liu, W. Liu, T. Mei, and H. Ma, “Provid: Progressive and multimodal vehicle reidentification for large-scale urban surveillance,” *IEEE Transactions on Multimedia*, vol. 20, no. 3, pp. 645–658, 2018. DOI: [10.1109/TMM.2017.2751966](https://doi.org/10.1109/TMM.2017.2751966).
- [199] G. Wang, J. Lai, P. Huang, and X. Xie, “Spatial-temporal person re-identification,” *ArXiv*, vol. abs/1812.03282, 2019.
- [200] X. Liu, P. Ghosh, O. Ulutan, B. S. Manjunath, K. Chan, and R. Govindan, “Caesar: Cross-camera complex activity recognition,” in *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, ser. SenSys ’19, New York, New York: Association for Computing Machinery, 2019, pp. 232–244, ISBN: 9781450369503. DOI: [10.1145/3356250.3360041](https://doi.org/10.1145/3356250.3360041). [Online]. Available: <https://doi.org/10.1145/3356250.3360041>.

- [201] Y. Lu, A. Chowdhery, and S. Kandula, “Optasia: A relational platform for efficient large-scale video analytics,” in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, ser. SoCC ’16, Santa Clara, CA, USA: Association for Computing Machinery, 2016, pp. 57–70, ISBN: 9781450345255. DOI: [10.1145/2987550.2987564](https://doi.org/10.1145/2987550.2987564). [Online]. Available: <https://doi.org/10.1145/2987550.2987564>.
- [202] J. Jiang, Y. Zhou, G. Ananthanarayanan, Y. Shu, and A. A. Chien, “Networked cameras are the new big data clusters,” in *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, ser. HotEdgeVideo’19, Los Cabos, Mexico: Association for Computing Machinery, 2019, pp. 1–7, ISBN: 9781450369282. DOI: [10.1145/3349614.3356026](https://doi.org/10.1145/3349614.3356026). [Online]. Available: <https://doi.org/10.1145/3349614.3356026>.
- [203] P. Liu, B. Qi, and S. Banerjee, “Edgeeye: An edge service framework for real-time intelligent video analytics,” in *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking*, ser. EdgeSys’18, Munich, Germany: ACM, 2018, pp. 1–6, ISBN: 978-1-4503-5837-8. DOI: [10.1145/3213344.3213345](https://doi.org/10.1145/3213344.3213345). [Online]. Available: <http://doi.acm.org/10.1145/3213344.3213345>.
- [204] A. Ravindran and A. George, “An edge datastore architecture for latency-critical distributed machine vision applications,” in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, Boston, MA: USENIX Association, Jul. 2018. [Online]. Available: <https://www.usenix.org/conference/hotedge18/presentation/ravindran>.
- [205] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, “Cachier: Edge-caching for recognition applications,” in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 276–286. DOI: [10.1109/ICDCS.2017.94](https://doi.org/10.1109/ICDCS.2017.94).
- [206] W. Magdy and G. J. Jones, “Pres: A score metric for evaluating recall-oriented information retrieval applications,” in *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR ’10, Geneva, Switzerland: Association for Computing Machinery, 2010, pp. 611–618, ISBN: 9781450301534. DOI: [10.1145/1835449.1835551](https://doi.org/10.1145/1835449.1835551). [Online]. Available: <https://doi.org/10.1145/1835449.1835551>.

VITA

Tiantu Xu was born in Beijing, China. He received his bachelor's degree (B.S.) in Applied Physics from the University of Science and Technology of China (USTC) in 2016. In the same year, he started to pursue his Ph.D. degree under the guidance of Prof. Felix Xiaozhu Lin at the School of Electrical and Computer Engineering, Purdue University. His research focuses on building software systems for large-scale retrospective video analytics related to computer vision workloads.