VStore: A Data Store for Analytics on Large Videos

Tiantu Xu
Purdue ECE

Luis Materon Botelho
Purdue ECE

Felix Xiaozhu Lin
Purdue ECE

Abstract

We present VStore, a data store for supporting fast, resource-efficient analytics over large archival videos. VStore manages video ingestion, storage, retrieval, and consumption. It controls video formats along the video data path. It is challenged by i) the huge combinatorial space of video format knobs; ii) the complex impacts of these knobs and their high profiling cost; iii) optimizing for multiple resource types. It explores an idea called 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. In this process, VStore derives an optimal set of video formats, optimizing for different resources in a progressive manner.

VStore automatically derives large, complex configurations consisting of more than one hundred knobs over tens of video formats. In response to queries, VStore selects video formats catering to the executed operators and the target accuracy. It streams video data from disks through decoder to operators. It runs queries as fast as 362× of video realtime.

CCS Concepts • **Information systems → Data analytics**; • **Computing methodologies → Computer vision tasks**; *Object recognition*;

Keywords Video Analytics, Data Store, Deep Neural Networks

ACM Reference Format:
Tiantu Xu, Luis Materon Botelho, and Felix Xiaozhu Lin. 2019. VStore: A Data Store for Analytics on Large Videos. In *Fourteenth EuroSys Conference 2019 (EuroSys '19), March 25–28, 2019, Dresden, Germany.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3302424.3303971

1 Introduction

Pervasive cameras produce videos at an unprecedented rate. Over the past 10 years, the annual shipments of surveillance cameras grow by 10×, to 130M per year [28]. Many campuses are reported to run more than 200 cameras 24×7 [57]. In such deployment, a single camera produces as much as 24 GB encoded video footage per day (720p at 30 fps).

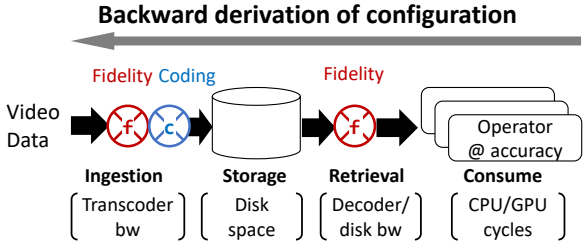


Figure 1. The VStore architecture, showing the video data path and backward derivation of configuration.

Retrospective video analytics To generate insights from enormous video data, retrospective analytics is vital: video streams are captured and stored on disks for a user-defined lifespan; users run queries over the stored videos on demand. Retrospective analytics offers several key advantages that live analytics lacks. i) Analyzing many video streams in real time is expensive, e.g., running deep neural networks over live videos from a $200 camera may require a $4000 GPU [34]. ii) Query types may only become known after the video capture [33]. iii) At query time, users may interactively revise their query types or parameters [19, 33], which may not be foreseen at ingestion time. iv) In many applications, only a small fraction of the video will be eventually queried [27], making live analytics an overkill.

A video query (e.g., "what are the license plate numbers of all blue cars in the last week?") is typically executed as a cascade of operators [32–34, 60, 67]. Given a query, a query engine assembles a cascade and run the operators. Query engines typically expose to users the trade-offs between operator accuracy and resource costs, allowing users to obtain inaccurate results with a shorter wait. The users thus can explore large videos interactively [33, 34]. Recent query engines show promise of high speed, e.g., consuming one-day video in several minutes [34].

Need for a video store While recent query engines assume *all* input data as raw frames present in memory, there lacks a video store that manages large videos for analytics. The store should orchestrate four major stages on the video data path: ingestion, storage, retrieval, and consumption, as shown in Figure 1. The four stages demand multiple hardware resources, including encoder/decoder bandwidth, disk space, and CPU/GPU cycles for query execution. The

EuroSys '19, March 25–28, 2019, Dresden, Germany

ACM ISBN 978-1-4503-6281-8/19/03…$15.00
https://doi.org/10.1145/3302424.3303971

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 [35]: they were designed for *human* consumers watching videos at 1×–2× speed of video realtime; they are incapable of serving some *algorithmic* consumers, i.e., operators, processing videos at more than 1000× video realtime. Shifting part of the query to ingestion [24] has important limitations and does not obviate the need for such a video store, as we will show in the paper.

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 [35]; while existing query engines recognize the significance of video formats [32, 33, 67] 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. 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.

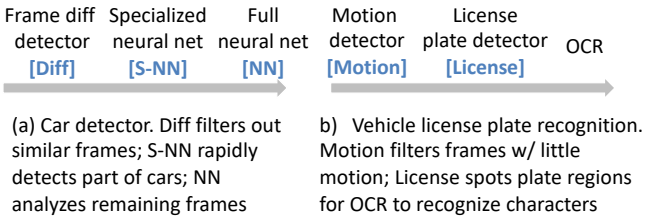


Figure 2. Video queries as operator cascades [34, 46].

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× 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× higher compared to systems that lack automatic configuration of video formats. VStore reduces the total configuration overhead by 5×.

Contributions We have made the following contributions.

- 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.

2 Motivations

2.1 Retrospective Video analytics

Query & operators A video query is typically executed as a cascade of operators. As shown in Figure 2, early operators scan most of the queried video timespan 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 [34]; 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 [32]. At runtime, an operator's target accuracy is set in queries [32, 33, 67]. VStore seeks to provision minimum resources for operators to achieve the target accuracy.

2.2 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 7.

• **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 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 [6], videos have age-based values. A store typically holds video footage for a user-defined lifespan [54]. 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 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. 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.

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

Equivalent FFmpeg options:
* CRF = 50, 40, 23, 0 **preset = veryslow, medium, veryfast, superfast, ultrafast

Table 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.

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 [24]. This approach has important limitations.

• It bakes query types in the ingestion. Video queries and operators are increasingly rich [15, 21, 42, 56, 64]; one operator (e.g., neural networks) may be instantiated with different parameters depending on training data [18]. 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 [19, 33].

• It prepays computation cost for all ingested videos. In many scenarios such as surveillance, only a small fraction of ingested video is eventually queried [18, 57]. As a result, most operator execution at ingestion is in vain.

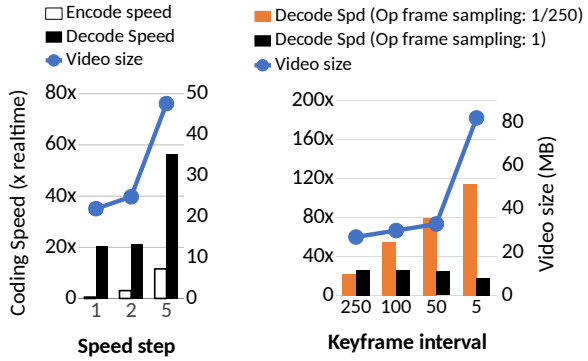
In comparison, by preparing data for queries, a video store supports richer query types, incurs lower ingestion cost, and allows flexible query-time trade-offs. Section 7 will provide further discussion.

2.3 Video Format Knobs

The video format is controlled by a set of parameters, or knobs. Table 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* $\vec{f}$. 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 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



(a) Coding can be speeded up at the expense of increased video size

b) Smaller intervals allows to skip more frames in decoding, if operators only sample a fraction of decoded frames to consume. Note: encoding speed mostly unaffected

Figure 3. Impacts of coding knobs. Video: 100 seconds from *tucson*. See Section 6 for dataset and test hardware.

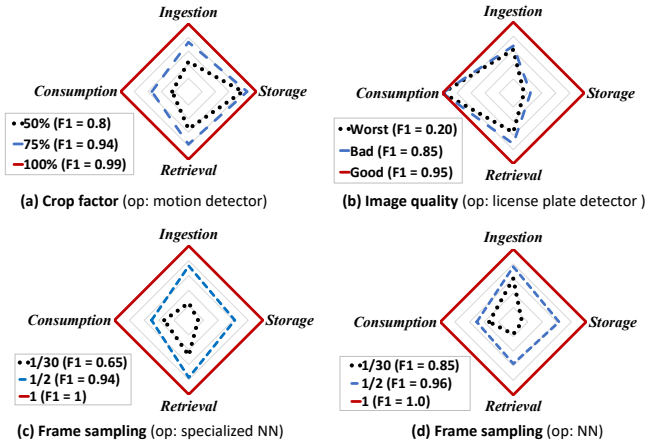


Figure 4. Fidelity knobs have high, complex impacts on costs of multiple components (normalized on each axis) and operator accuracy (annotated in legends). Each plot: one knob changing; all others fixed. See Section 6 for methodology.

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 [63]. 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 1 summarizes these knobs and Figure 3 shows their impacts. ***Speed step*** accelerates encoding/decoding at the expense of increased video size. As shown in Figure 3(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" [20]): 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(b), smaller keyframe intervals increase decoding speed by up to 6× 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 (see Section 3).

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

2.4 Knob impacts

As illustrated in Figure 1: 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. **Fidelity knobs enable rich cost/accuracy trade-offs.** As shown in Figure 4, one may reduce resource costs by up to 50% with minor (5%) accuracy loss. **The knobs enable rich trade-offs among resource types.** This is exemplified in Figure 5: 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. **Each knob has significant impacts.** Take Figure 4(b) as an example: one step change to image quality reduces accuracy from 0.95 to 0.85, the storage cost by 5×, and the ingestion cost by 40%. **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× 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 [67]. ii) The quantitative relations vary across operators and across video contents [32]. This is exemplified by Figure 4 (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

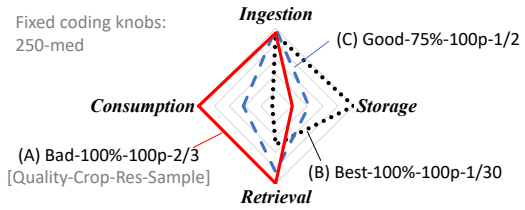


Figure 5. Disparate costs of fidelity options A–C, despite all leading to operator accuracy ≈ 0.8. Operator: License. Cost normalized on each axis. See Section 6 for methodology.

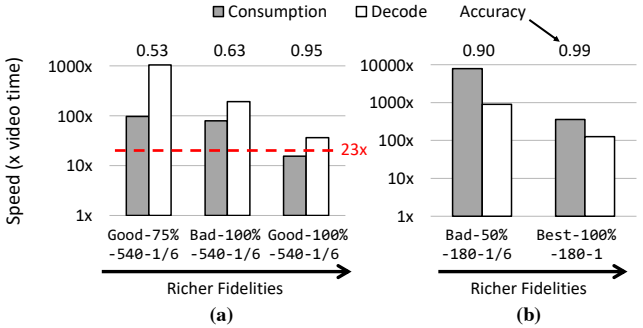


Figure 6. Video retrieval *could* bottleneck consumption. This is exemplified by the decoding speed vs. consumption speed comparisons for two different operators. **(a)** Operator: License. Consumption can be faster than decoding (speed shown as the dashed line), if the on-disk video is stored with the 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 6 for test hardware.

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 0.76 to 0.52. While prior work assumes that certain knobs have independent impacts on accuracy [32], 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 [32, 33, 67]. 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 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.

3.1 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. **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 6 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 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

Op	Description
Diff	Difference detector that detects frame differences [34]
S-NN	Specialized NN to detect a specific object [34]
NN	Generic Neural Networks, e.g., YOLO [53]
Motion	Motion detector using background subtraction [46]
License	License plate detector [46]
OCR	Optical character recognition [46]
Opflow	Optical flows for tracking object movements [48]
Color	Detector for contents of a specific color [33]
Contour	Detector for contour boundaries [47]

Table 2. The library of operators in the current VStore.

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* [3]. 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 6.

Layered encoding cannot simplify the problem Layered encoding promises space efficiency: it stores one video's multiple fidelity options as complementary layers [59]. However, layered encoding has important caveats. i) Each additional layer has non-trivial storage overhead (sometimes 40%–100%) [37] 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.

4 The VStore Design

4.1 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 2 listed 9 operators that are supported by the current VStore prototype. VStore tracks the whole set of <operator, accuracy> 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

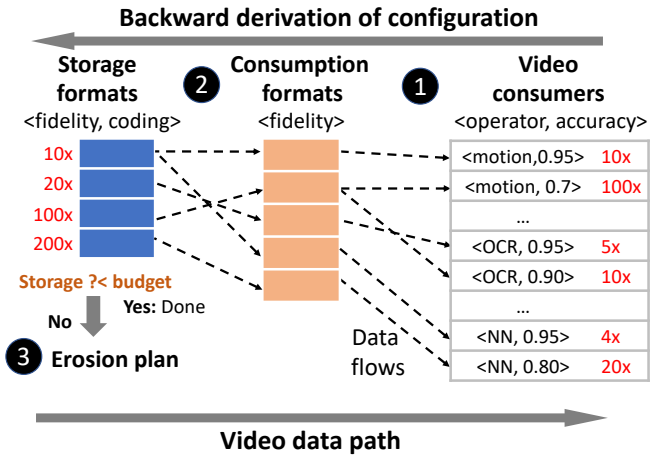


Figure 7. VStore derives the configuration of video formats. Example consumption/retrieval speed is shown.

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}. ii) Exhaustive profiling is expensive, as will be discussed in Section 4.2. iii) Optimizing for multiple resource types further complicates the problem.

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 [32, 67]. 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 7 ❶–❸. In this backward derivation, VStore optimizes for different resources in a progressive manner.

❶ Section 4.2: 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.

❷ Section 4.3: 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 7). 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 7). In this step, VStore optimizes for storage cost and keeps ingestion cost under budget.
❸ Section 4.4: 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 i) 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. ii) VStore does not manage algorithmic knobs internal to operators [32, 67]; doing so would allow new, useful trade-offs for consumption but not for ingestion, storage, or retrieval.

4.2 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 4 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 4(b).

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 × resolution × sampling rate), VStore searches for fidelity f_0' that leads to adequate accuracy and the lowest consumption cost. iii) As shown in Figure 8, VStore partitions the 3D space into a set of 2D spaces for search. To minimize the number of 2D

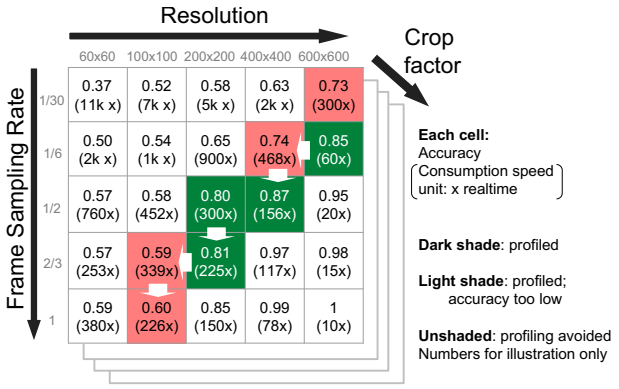


Figure 8. Search in a set of 2D spaces for a fidelity option with accuracy ≥ 0.8 and max consumption speed (i.e., min consumption cost).

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 × sampling rate), as illustrated in Figure 8. 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 8, 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 [16]. 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

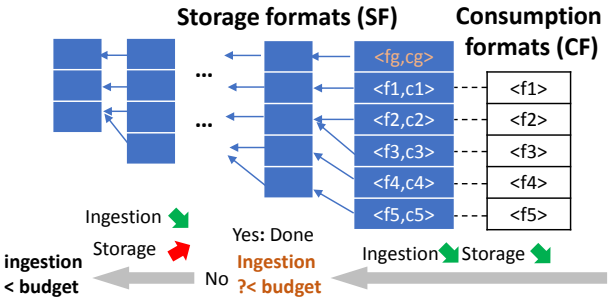


Figure 9. Iterative coalescing of storage formats.

for the same operator's different accuracies, VStore memoizes profiling results. Our evaluation (Section 6) 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 [16, 40].

4.3 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 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 [10, 11].

Algorithm sketch VStore coalesces the set of storage formats iteratively. Show on the right side of Figure 9, 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<fg,cg>: 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 4.4. All these SFs participate in coalescing.

How to coalesce a pair? VStore runs multiple rounds of pairwise coalescing. To coalesce SF0⟨f0,c0⟩ and SF1⟨f1,c1⟩ into SF2⟨f2,c2⟩, VStore picks f2 to be the knob-wise maximum of f0 and f1 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 SF2 with richer fidelity tends to be slower to decode than SF0/SF1. VStore therefore picks a cheaper coding option (c2) for SF2, so that decoding SF2 is fast enough for all previous consumers of SF0/SF1. Even if the cheapest coding option is not fast enough, VStore bypasses coding and stores raw frames for SF2. 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 9 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 $\min(O(N), |\mathbb{F}\times\mathbb{C}|)$.

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 $\min(O(N^3), |\mathbb{F}\times\mathbb{C}|)$. 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 Section 6. Furthermore, Section 6 will show that heuristic-based selection produces much more compact SFs.

4.4 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.

Our insight is as follows. As video content ages, the system may slowly give up some of the formats, freeing space

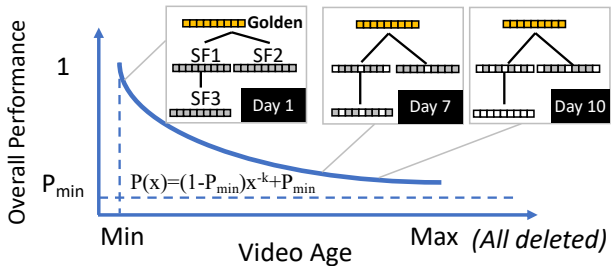


Figure 10. Data erosion decays operator speed and keeps storage cost under budget. Small cells: video segments.

by relaxing the requirement for adequate retrieving speed on aged data (Section 3, 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 10, 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 10. 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 [12], 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 [6]. 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 [43]. 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.

Putting it together 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.

5 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 [29], 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 [46] 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 [34] is a recent research engine. It combines operators that execute at various speeds and invoke deep NN. It invokes TensorFlow [5] as the NN framework, which runs on GPU. **Operator lib:** The two query engines provide 6 operators as shown in Figure 2. In particular, S-NN uses a very shallow AlexNet [38] produced by NoScope's model search and NN uses YOLOv2 [53].

6 Evaluation

We answer the following questions in evaluation:

§6.2 Does VStore provide good end-to-end results?
§6.3 Does VStore adapt configurations to resource budgets?
§6.4 Does VStore incur low overhead in configuration?

6.1 Methodology

Video Datasets We carried out our evaluation on six videos, extensively used as benchmarks in prior work [24, 32–34]. 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 admin, declare a set of accuracy levels {0.95, 0.9, 0.8, 0.7} for each operator. These accuracies are typical in prior work [32]. 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 2: 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 [34, 46]. 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 [32]. 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.

6.2 End-to-end Results

Configuration by VStore VStore automatically configuring video formats based on its profiling. Table 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(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 2.4. 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(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 4.3). 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 4.3: 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→1** stores videos in the golden format (SF_g in Table 3). All consumers consume videos in this golden format. This resembles a video database oblivious to algorithmic consumers.
- **1→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.

	Diff	S-NN	NN	Motion	License	OCR
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
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
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
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

(a) Consumption Formats (CF)

	Storage Formats (SFs)
SFg	best-720p-1-100% 250-slowest 1393KB 23x
SF1	good-540p-1/6-100% 250-slowest 409KB 178x
SF2	best-540p-1/30-100% 10-fast 92KB 331x
SF3	best-200p-1-100% RAW 1843KB[1] 1137-34132x[2]

(b) Storage Formats (SF)

(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.

Table 3. A sample configuration of video formats automatically derived by VStore.

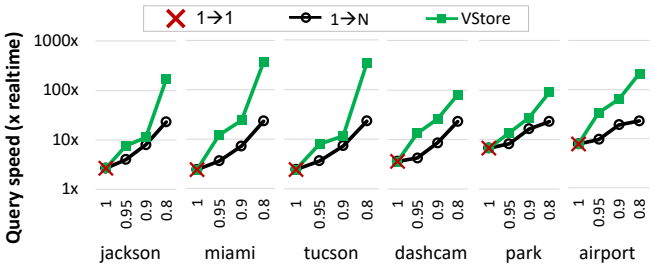
• **N→N** stores videos in 21 SFs, one for each unique CF. This is equivalent to VStore giving up its coalescing of SFs.

Query speed As shown in Figure 11(a), VStore achieves good query speed overall, up to 362× realtime. VStore's speed is incomparable with performance reported for retrospective analytics engines [33, 34]: 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×. 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.

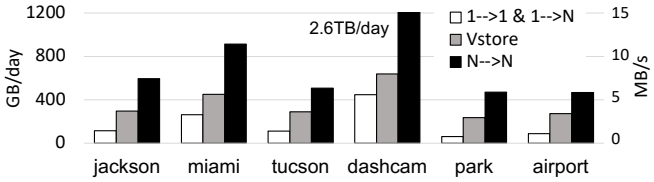
Figure 11(a) also shows the query speed under alternative configurations. 1→1 achieves the best accuracy (treated as the ground truth) as it consumes video in the full fidelity as ingested. However, it cannot exploit accuracy/cost trade-offs, offering a fixed operating point. By contrast, VStore offers 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(a). As a result, VStore outperforms 1→N by 3×-16×, demonstrating the necessity of the SF set.

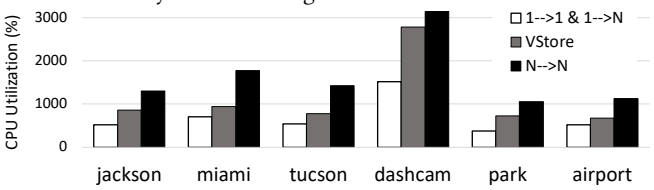
Storage cost Figure 11(b) compares the storage costs. Among all, N→N incurs the highest costs, because it stores 21 video versions in total. For *dashcam*, a video stream with intensive motion which makes video coding less effective, the storage cost reaches as high as 2.6 TB/day, filling a 10TB hard drive



(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 coalsecing 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 4 for more.

Figure 11. End-to-end result.

		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-slow	250-med	250-fast
	SF1	250-slowest	250-slow	250-slow	250-med	250-fast
	SF2	10-fast	10-fast	10-fast	10-fast	250-fast
	SF3	RAW	RAW	RAW	RAW	RAW

Coding option: "Keyframe Interval" - "SpeedStep"

Table 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.

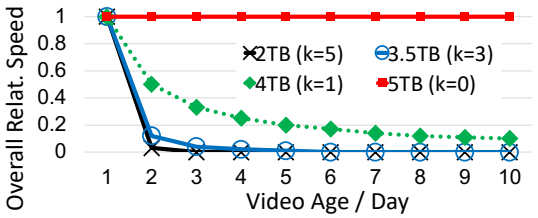
in four days. In comparison, VStore consolidates the storage formats effectively and therefore reduces the storage cost by 2×-5×. 1→1 and 1→N require the lowest storage space as they only save one video version per ingestion stream; yet, they suffer from high retrieval cost and low query speed.

Ingestion cost Figure 11(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.

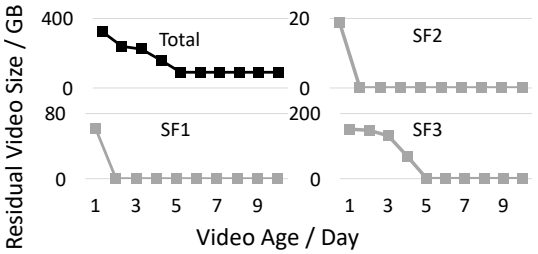
Overall, on most videos VStore requires around 9 cores to ingest one video stream, transcoding it into the 4 SFs in real time (30 fps). Ingesting *dashcam* is much more expensive, as the video contains intensive motion. VStore's cost is 30%–50% lower than N→N, which must transcode each stream to 21 SFs. 1→1 and 1→N incur the lowest ingestion cost as they only transcode the ingestion stream to the golden format, yet at the expense of costly retrieval and slow query speed.

6.3 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 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).



(a) Operator speed decays as the video ages. For lower storage budget, VStore chooses more aggressive decay (higher k)



(b) Storage cost decreases as the video ages. Storage budget set to 2TB. 3 stored versions and the total size are shown. The total size further includes the golden format (not shown), which is not eroded by design.

Figure 13. Age-based decay in operator speed (a) and reducing storage cost (b) to respect storage budget.

We also find out that SFs as well as the ingestion cost quickly plateaus as VStore's library includes more operators. Figure 12 shows how the ingestion cost increases as operators are sequentially added, following the order listed in Table 2, to VStore's library. The ingestion cost stabilizes as the number of operators exceeds 5, as additional operators share existing SFs.

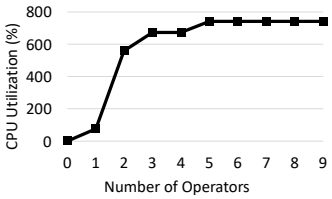


Figure 12. Transcoding cost does not scale up with the number of operators. Operator sequence follows Table 2.

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(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 13(a). Further reducing the storage budget prompts data erosion. With a 4 TB budget, VStore decays the overall operator speed (defined in Section 4.4) following a power law function (k=1). As we further reduce the budget, VStore plans more aggressive decays to respect the budget. Figure 13(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

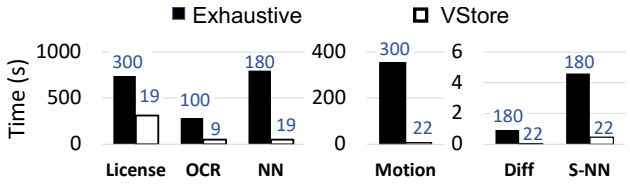


Figure 14. Time spent on deriving consumption formats. Numbers of profiling runs are annotated above columns. Each required profiling runs on a 10-second video segment. VStore reduces overhead by 5× in total.

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).

6.4 Configuration Overhead

VStore incurs moderate configuration overhead, thanks to our techniques in Section 4.2 and 4.3. Overall, one complete configuration (including all required profiling) takes around 500 seconds, suggesting the store can afford one configuration process in about every 1 hour online.

Configuring consumption formats Figure 14 shows the overhead in determining consumption formats. Compared to exhaustive profiling of all fidelity options, VStore reduces the number of profiling runs by 9×–15× and the total profiling delay by 5×, 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 4.3). We compare it to exhaustive enumeration, on deriving SFs from the 12 CFs 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× 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× higher storage cost. This is because the latter strategy, while simple, overlooks the fact that different knobs have complex and varying resource impacts (Section 2.4), which cannot be simply normalized across knobs.

7 Discussion

Adapting to changes in operators and hardware VStore works with any possible queries composed by operators/accuracies pre-defined in its library (Section 2.2). 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 3.1). 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 [24] As stated in Section 2.2, 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 12, 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) [41, 66]. 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 [1].

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, $\alpha = 1/48$ [24].

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.

8 Related Work

Optimizing video analytics Catering to retrospective video analytics, BlazeIt [33] proposes a query model and corresponding execution techniques [33]. NoScope [34] reduces query cost with cheap early filters before expensive NN. To run NNs on mobile devices, MCDNN [23] trades off between accuracy and resource constraints by model compression.

Optimizing live video analytics For distributed, live video analytics, VideoStorm [67] and VideoEdge [26] search for best knobs and query placements over clusters to meet accuracy/delay requirements. For live video analytics on the edge, LAVEA[65] and Vigil [68] partitions analytics pipelines between the edge and the cloud. Jain *et al.* [31] optimize video analytics over multiple cameras through cross-camera correlations. Pakha *et al.* [49] 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 [9] accelerates photo access through metadata lookups in main memory. Intel's VDMS [22, 55] accelerates image data access through a combination of graph-based metadata and array-based images backed by TileDB [50]. They focus on images rather than videos. Targeting NN training, NVIDIA's Video Loader [14] (a wrapper over NVDEC and FFmpeg) optimizes random loads of encoded video frames. To support video analytics at scale, Scanner [51] 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 [2] 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 [6, 7]. 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 [30, 61] 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 [58] and streaming databases [8, 44] enable sharing data and computation across queries with techniques such as scan sharing [39, 52, 69]. 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 [4, 13, 62], 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 [17, 36]. Since then, Kang *et al.* [35] optimizes placement of on-disk video layers in order to reduce disk seek. Oh *et al.* [45] segments videos into shots, which are easier for humans to browse and search. Recently, SVE [25] is a distributed service for fast transcoding of uploaded videos in datacenters. ExCamera [20] 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.

9 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× of video realtime.

Acknowledgments

The authors were supported in part by NSF Award 1619075 (including REU) and a Google Faculty Award. The authors thank the anonymous reviewers for their feedback and Dr. Johannes Gehrke for shepherding. The authors thank NVIDIA for their GPU donation.

References

[1] 2018. Amazon EC2 P3 Instances. https://aws.amazon.com/ec2/instance-types/p3/.

[2] 2018. NVIDIA. https://developer.nvidia.com/deepstream-sdk.

[3] 2018. RollingDB Storage Library. https://github.com/openalpr/rollingdb.

[4] Daniel J. Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. 2003. Aurora: A New Model and Architecture for Data Stream Management. *The VLDB Journal* 12, 2 (Aug. 2003), 120–139. https://doi.org/10.1007/s00778-003-0095-z

[5] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 265–283. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[6] Nitin Agrawal and Ashish Vulimiri. 2017. Low-Latency Analytics on Colossal Data Streams with SummaryStore. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. ACM, New York, NY, USA, 647–664. https://doi.org/10.1145/3132747.3132758

[7] Michael P Andersen and David E. Culler. 2016. BTrDB: Optimizing Storage System Design for Timeseries Processing. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*. USENIX Association, Santa Clara, CA, 39–52. https://www.usenix.org/conference/fast16/technical-sessions/presentation/andersen

[8] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. 2002. Models and Issues in Data Stream Systems. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '02)*. ACM, New York, NY, USA, 1–16. https://doi.org/10.1145/543613.543615

[9] Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, and Peter Vajgel. 2010. Finding a Needle in Haystack: Facebook's Photo Storage. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*. USENIX Association, Berkeley, CA, USA, 47–60. http://dl.acm.org/citation.cfm?id=1924943.1924947

[10] E. T. Bell. 1934. Exponential Numbers. *The American Mathematical Monthly* 41, 7 (1934), 411–419. http://www.jstor.org/stable/2300300

[11] E. T. Bell. 1934. Exponential Polynomials. *Annals of Mathematics* 35, 2 (1934), 258–277. http://www.jstor.org/stable/1968431

[12] Dimitri Bertsekas and Robert Gallager. 1992. *Data Networks (2Nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[13] Don Carney, Uğur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. 2002. Monitoring Streams: A New Class of Data Management Applications. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB '02)*. VLDB Endowment, 215–226. http://dl.acm.org/citation.cfm?id=1287369.1287389

[14] Jared Casper, Jon Barker, and Bryan Catanzaro. 2018. NVVL: NVIDIA Video Loader. https://github.com/NVIDIA/nvvl.

[15] Y. Chen, X. Zhu, W. Zheng, and J. Lai. 2018. Person Re-Identification by Camera Correlation Aware Feature Augmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40, 2 (Feb 2018), 392–408. https://doi.org/10.1109/TPAMI.2017.2666805

[16] Yongxi Cheng, Xiaoming Sun, and Yiqun Lisa Yin. 2008. Searching monotone multi-dimensional arrays. *Discrete Mathematics* 308, 11 (2008), 2213–2221.

[17] Tzi-cker Chiueh and Randy H. Katz. 1993. Multi-resolution Video Representation for Parallel Disk Arrays. In *Proceedings of the First ACM International Conference on Multimedia (MULTIMEDIA '93)*. ACM, New York, NY, USA, 401–409. https://doi.org/10.1145/166266.168438

[18] Ziqiang Feng, Shilpa George, Jan Harkes, Padmanabhan Pillai, Roberta Klatzky, and Mahadev Satyanarayanan. 2019. Eureka: Edge-based Discovery of Training Data for Machine Learning. *IEEE Internet Computing* PP (01 2019), 1–1. https://doi.org/10.1109/MIC.2019.2892941

[19] Ziqiang Feng, Junjue Wang, Jan Harkes, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2018. EVA: An Efficient System for Exploratory Video Analysis. *SysML* (2018).

[20] Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. 2017. 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)*. USENIX Association, Boston, MA, 363–376. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi

[21] Ross Girshick. 2015. Fast R-CNN. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV) (ICCV '15)*. IEEE Computer Society, Washington, DC, USA, 1440–1448. https://doi.org/10.1109/ICCV.2015.169

[22] Vishakha Gupta-Cledat, Luis Remis, and Christina R Strong. 2017. Addressing the Dark Side of Vision Research: Storage. In *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17)*. USENIX Association, Santa Clara, CA. https://www.usenix.org/conference/hotstorage17/program/presentation/gupta-cledat

[23] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. 2016. 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 (MobiSys '16)*. ACM, New York, NY, USA, 123–136. https://doi.org/10.1145/2906388.2906396

[24] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B. Gibbons, and Onur Mutlu. 2018. Focus: Querying Large Video Datasets with Low Latency and Low Cost. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA. https://www.usenix.org/conference/osdi18/presentation/hsieh

[25] Qi Huang, Petchean Ang, Peter Knowles, Tomasz Nykiel, Iaroslav Tverdokhlib, Amit Yajurvedi, Paul Dapolito, IV, Xifan Yan, Maxim Bykov, Chuen Liang, Mohit Talwar, Abhishek Mathur, Sachin Kulkarni, Matthew Burke, and Wyatt Lloyd. 2017. SVE: Distributed Video Processing at Facebook Scale. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. ACM, New York, NY, USA, 87–103. https://doi.org/10.1145/3132747.3132775

[26] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter BodÃŋk, Leana Golubchik, Minlan Yu, Victor Bahl, and Matthai Philipose. 2018. VideoEdge: Processing Camera Streams using Hierarchical Clusters. https://www.microsoft.com/en-us/research/publication/videoedge-processing-camera-streams-using-hierarchical-clusters/

[27] IHS. 2016. Top Video Surveillance Trends for 2016.

[28] IHS. 2018. Top Video Surveillance Trends for 2018.

[29] iMatix Corporation. 2018. Lightning Memory-mapped Database. https://symas.com/lmdb/.

[30] InfluxData. 2018. InfluxDB. https://www.influxdata.com/.

[31] Samvit Jain, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, and Joseph E Gonzalez. 2018. Scaling Video Analytics Systems to Large Camera Deployments. *arXiv preprint arXiv:1809.02318* (2018).

[32] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: Scalable Adaptation of Video Analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. ACM, New York, NY, USA, 253–266. https://doi.org/10.1145/3230543.3230574

[33] Daniel Kang, Peter Bailis, and Matei Zaharia. 2018. BlazeIt: Fast Exploratory Video Queries using Neural Networks. *arXiv preprint arXiv:1805.01046* (2018).

[34] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Neural Network Queries over Video at Scale. *Proc. VLDB Endow.* 10, 11 (Aug. 2017), 1586–1597. https://doi.org/10.14778/3137628.3137664

[35] Sooyong Kang, Sungwoo Hong, and Youjip Won. 2009. Storage technique for real-time streaming of layered video. *Multimedia Systems* 15, 2 (01 Apr 2009), 63–81. https://doi.org/10.1007/s00530-008-0147-8

[36] Kimberly Keeton and Randy H. Katz. 1995. Evaluating video layout strategies for a high-performance storage server. *Multimedia Systems* 3, 2 (01 May 1995), 43–52. https://doi.org/10.1007/BF01219800

[37] Christian Kreuzberger, Daniel Posch, and Hermann Hellwagner. 2015. A Scalable Video Coding Dataset and Toolchain for Dynamic Adaptive Streaming over HTTP. In *Proceedings of the 6th ACM Multimedia Systems Conference (MMSys '15)*. ACM, New York, NY, USA, 213–218. https://doi.org/10.1145/2713168.2713193

[38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. 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., 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[39] C. A. Lang, B. Bhattacharjee, T. Malkemus, S. Padmanabhan, and K. Wong. 2007. Increasing Buffer-Locality for Multiple Relational Table Scans through Grouping and Throttling. In *2007 IEEE 23rd International Conference on Data Engineering*. 1136–1145. https://doi.org/10.1109/ICDE.2007.368972

[40] Nathan Linial and Michael Saks. 1985. Searching ordered structures. *Journal of algorithms* 6, 1 (1985), 86–103.

[41] Peng Liu, Jongwon Yoon, Lance Johnson, and Suman Banerjee. 2016. Greening the Video Transcoding Service with Low-Cost Hardware Transcoders. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. USENIX Association, Denver, CO, 407–419. https://www.usenix.org/conference/atc16/technical-sessions/presentation/liu

[42] Shayan Modiri Assari, Haroon Idrees, and Mubarak Shah. 2016. *Human Re-identification in Crowd Videos Using Personal, Social and Environmental Constraints*. Springer International Publishing, Cham, 119–136. https://doi.org/10.1007/978-3-319-46475-6$_8$

[43] Ingo Molnár. 2007. [patch] Modular Scheduler Core and Completely Fair Scheduler. http://lwn.net/Articles/230501/.

[44] Rajeev Motwani, Jennifer Widom, Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Gurmeet Manku, Chris Olston, Justin Rosenstein, and Rohit Varma. 2003. Query Processing, Resource Management, and Approximation in a Data Stream Management System. In *IN CIDR*. 245–256.

[45] JungHwan Oh and Kien A. Hua. 2000. 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 (SIGMOD '00)*. ACM, New York, NY, USA, 415–426. https://doi.org/10.1145/342009.335436

[46] OpenALPR Technology, Inc. 2018. OpenALPR. https://github.com/openalpr/openalpr.

[47] OpenCV. 2018. Contours.

[48] OpenCV. 2018. Optical Flow.

[49] Chrisma Pakha, Aakanksha Chowdhery, and Junchen Jiang. 2018. Reinventing Video Streaming for Distributed Vision Analytics. In *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*. USENIX Association, Boston, MA. https://www.usenix.org/conference/hotcloud18/presentation/pakha

[50] Stavros Papadopoulos, Kushal Datta, Samuel Madden, and Timothy Mattson. 2016. The TileDB Array Data Storage Manager. *Proc. VLDB Endow.* 10, 4 (Nov. 2016), 349–360. https://doi.org/10.14778/3025111.3025117

[51] Alex Poms, Will Crichton, Pat Hanrahan, and Kayvon Fatahalian. 2018. Scanner: Efficient Video Analysis at Scale. *ACM Trans. Graph.* 37, 4, Article 138 (July 2018), 13 pages. https://doi.org/10.1145/3197517.3201394

[52] Lin Qiao, Vijayshankar Raman, Frederick Reiss, Peter J. Haas, and Guy M. Lohman. 2008. Main-memory Scan Sharing for Multi-core CPUs. *Proc. VLDB Endow.* 1, 1 (Aug. 2008), 610–621. https://doi.org/10.14778/1453856.1453924

[53] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 779–788. https://doi.org/10.1109/CVPR.2016.91

[54] Oracle. 2015. Dramatically Reduce the Cost and Complexity of Video Surveillance Storage. https://www.oracle.com/assets/wp-video-surveillance-storage-2288409.pdf.

[55] Luis Remis, Vishakha Gupta-Cledat, Christina R. Strong, and Margriet IJzerman-Korevaar. 2018. VDMS: An Efficient Big-Visual-Data Access for Machine Learning Workloads. *CoRR* abs/1810.11832 (2018).

[56] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. 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 (NIPS'15)*. MIT Press, Cambridge, MA, USA, 91–99. http://dl.acm.org/citation.cfm?id=2969239.2969250

[57] Seagate. 2017. 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.

[58] Timos K. Sellis. 1988. Multiple-query Optimization. *ACM Trans. Database Syst.* 13, 1 (March 1988), 23–52. https://doi.org/10.1145/42201.42203

[59] Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hossfeld, and Phuoc Tran-Gia. 2015. A survey on quality of experience of HTTP adaptive streaming. *IEEE Communications Surveys & Tutorials* 17, 1 (2015), 469–492.

[60] Haichen Shen, Seungyeop Han, Matthai Philipose, and Arvind Krishnamurthy. 2017. Fast Video Classification via Adaptive Cascading of Deep Models. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[61] V. Srinivasan, Brian Bulkowski, Wei-Ling Chu, Sunil Sayyaparaju, Andrew Gooding, Rajkumar Iyer, Ashish Shinde, and Thomas Lopatic. 2016. Aerospike: Architecture of a Real-time Operational DBMS. *Proc. VLDB Endow.* 9, 13 (Sept. 2016), 1389–1400. https://doi.org/10.14778/3007263.3007276

[62] Nesime Tatbul, Uğur Çetintemel, Stan Zdonik, Mitch Cherniack, and Michael Stonebraker. 2003. Load Shedding in a Data Stream Manager. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29 (VLDB '03)*. VLDB Endowment, 309–320. http://dl.acm.org/citation.cfm?id=1315451.1315479

[63] Chao-Yuan Wu, Manzil Zaheer, Hexiang Hu, R. Manmatha, Alexander J. Smola, and Philipp Krähenbühl. 2018. Compressed Video Action Recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[64] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. 2013. Online Object Tracking: A Benchmark. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '13)*. IEEE Computer Society, Washington, DC, USA, 2411–2418. https://doi.org/10.1109/CVPR.2013.312

[65] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li. 2017. LAVEA: Latency-Aware Video Analytics on Edge Computing Platform. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2573–2574. https://doi.org/10.1109/ICDCS.2017.182

[66] J. Yoon, P. Liu, and S. Banerjee. 2016. Low-Cost Video Transcoding at the Wireless Edge. In *2016 IEEE/ACM Symposium on Edge Computing (SEC)*. 129–141. https://doi.org/10.1109/SEC.2016.8

[67] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 377–392. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zhang

[68] Tan Zhang, Aakanksha Chowdhery, Paramvir (Victor) Bahl, Kyle Jamieson, and Suman Banerjee. 2015. The Design and Implementation of a Wireless Video Surveillance System. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom '15)*. ACM, New York, NY, USA, 426–438. https://doi.org/10.1145/2789168.2790123

[69] Marcin Zukowski, Sándor Héman, Niels Nes, and Peter Boncz. 2007. Cooperative Scans: Dynamic Bandwidth Sharing in a DBMS. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB '07)*. VLDB Endowment, 723–734. http://dl.acm.org/citation.cfm?id=1325851.1325934