Video-zilla: An Indexing Layer for Large-Scale Video Analytics

Bo Hu
b.hu@yale.edu
Yale University

Peizhen Guo
patrick.guopz@gmail.com
Yale University

Wenjun Hu
wenjun.hu@yale.edu
Yale University

Abstract

Pervasive deployment of surveillance cameras today poses enormous scalability challenges to video analytics systems operating over many camera feeds. Currently, there are few indexing tools to organize video feeds beyond what is provided by a standard file system. Recent video analytic systems implement application-specific frame profiling and sampling techniques to reduce the number of raw videos processed, leveraging frame-level redundancy or manually labeled spatial-temporal correlation between cameras.

This paper presents *Video-zilla*, a standalone indexing layer between video query systems and a video store to organize video data. We propose a video data unit abstraction, *semantic video stream (SVS)*, based on a notion of distance between objects in the video. SVS implicitly captures *scenes*, which is missing from current video content characterization and a middle ground between individual frames and an entire camera feed. We then build a hierarchical index that exposes the semantic similarity both within and across camera feeds, such that *Video-zilla* can quickly cluster video feeds based on their content semantics without manual labeling. We implement and evaluate *Video-zilla* in three use cases: object identification queries, clustering for training specialized DNNs, and archival services. In all three cases, *Video-zilla* reduces the time complexity of inter-camera video analytics from linear with the number of cameras to sublinear, and reduces query resource usage by up to 14× compared to using frame-level or spatial-temporal similarity built into existing query systems.

CCS Concepts

• **Information systems → Data management systems**.

Keywords

edge computing, video analytics

ACM Reference Format:
Bo Hu, Peizhen Guo, and Wenjun Hu. 2022. *Video-zilla*: An Indexing Layer for Large-Scale Video Analytics. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3514221.3517840

1 Introduction

Recent years have witnessed a sharp uptick of the number of video cameras. 2018 alone saw 140 million shipments of video cameras [41]. These are widely deployed for monitoring and surveillance in diverse sectors, ranging from traffic control [90], crime investigation [27], to healthcare [91]. For example, the city of Bellevue, WA has deployed cameras at many intersections to monitor traffic; According to the British Security Industry Authority [11], approximately 300,000 cameras are already deployed in UK schools. These surveillance applications have generated a wide variety of video analytics workloads. At the core of these workloads is a set of computer vision tasks, such as object detection and tracking, which then form the basis for specific applications such as license plate identification, tracking suspects, and fall detection for elder care.

While video content analysis dates back decades, today's large-scale video analytics landscape is drastically different. First, video analytics used to be run on a small number of static video clips (i.e., pre-recorded with a fixed number of frames); Instead, we now have many surveillance cameras generating continuous video feeds. A single camera alone capturing video at 30 frames per second can generate 20 GB of video per day [82]. Second, the computer vision tasks have also grown in sophistication thanks to the increasing adoption of deep learning inference. Applying a state-of-the-art object detection DNN in real time (i.e., processing 30+ frames per second) to a single video feed requires a powerful GPU [45] costing $4000. Further, it is increasingly common to aggregate and analyze a large number of video feeds [42], for example, to track a suspect car around a city. The sheer volume of the videos today presents colossal scalability challenges to large-scale video analytics systems.

Existing large-scale video analytics systems tackle the daunting scalability prospects by leveraging the inherent redundancy within video feeds [37, 43, 45]. For example, objects in successive frames are likely to be the same, exhibiting *frame-level redundancy*; Cameras deployed at the same intersection often capture the same vehicles and pedestrians, though from different perspectives, due to the *spatial-temporal correlation* of camera placement and captured scenes. The former optimizes for individual frame retrieval, oblivious to the collective semantics of a feed, while the latter optimizes across video feeds, though requiring manual labeling. Redundancy elimination combines naturally with edge processing [88] and large-scale video analytics is a killer app for edge computing [42].

Still, neither redundancy management strategy can keep pace with the growing number of video feeds. Inter-feed analytics jobs effectively need to profile each camera feed individually. Fundamentally, a camera feed comprises of *scenes*, each carrying an implicit, collective sense of content semantics, such as *parking lot*, *downtown*, and *school*. This semantics dictates the type of objects and events featured in the feed. Exposing this semantics facilitates further optimizations to process the most relevant (subsets of) feeds. However, there are few mechanisms available to recognize this collective semantics and organize feeds. Feeds today are mainly stored in a file system following common encoding formats. Recent video analytic systems each implements application-specific frame profiling and sampling strategies to reduce redundant data processing. (Section 2)

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3517840

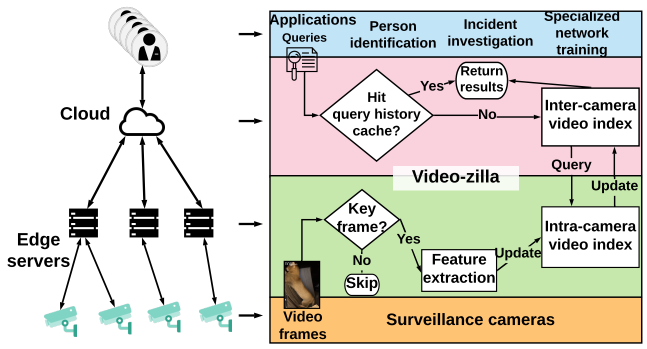


Figure 1: *Video-zilla* architecture

In this paper, we build *Video-zilla*, a standalone indexing layer interposed between video query systems and a video store to organize large-scale, multi-camera feeds. We propose a video data unit abstraction called *semantic video stream (SVS)* based on a notion of distance between objects in the video (Section 3). SVS implicitly characterizes the video content by *scenes*, which is missing from current video data abstractions and offers a suitable middle-ground granularity between individual frames and an entire camera feed. We then build a hierarchical index that analyzes and exposes the semantic similarity both within and across camera feeds (Section 4). This way, *Video-zilla* can quickly organize video feeds based on their content semantics without manual analysis of video content, while preserving the boundaries between cameras (Section 5). *Video-zilla* provides a generic service to a range of query systems. The index can be run either offline over static videos or online over live feeds with new frames arriving continuously. The built-in automatic video segmentation naturally delineates different scenes, which is also useful for moving cameras like dashcams or drones capturing frequent changing scenes. Finally, we refactor existing large-scale video analytics pipelines to separate out frame content analysis as an independent module, providing a per-camera video ingestion interface to the underlying surveillance cameras and a centralized query interface to upper-level video query engines (Section 6).

Video-zilla effectively acts as an *explanation database* [14, 69, 70, 85] for the video data, and a similar approach can apply to other unstructured but correlated IoT data. We evaluate *Video-zilla* with three case studies, object identification, specialized network training and video archiving (Section 7). Compared to existing indexing strategies, *Video-zilla* reduces query resource usage by up to 14× and can offer video clustering or archival support that is currently not feasible. *Video-zilla* can replace substantial code development for video content analysis in existing analytics systems with a few lines of query code.

In summary, we make three contributions: First, we propose Semantic Video Stream (SVS), a novel video data unit abstraction based on a notion of distance between objects in the video, which abstracts away the video content precisely and concisely. Second, based on SVS, we build a hierarchical video index as a data organization strategy to expose the correlation between and within camera feeds. By preserving the boundary between cameras while capturing their correlation, this lends to easy incorporation of additional cameras or policies such as privacy measures. Third, we advocate for refactoring current video analytic systems into a generic indexing layer, *Video-zilla*, and the specific analytics queries on top of that. *Video-zilla* can simplify building analytic applications and reduce time complexity of inter-camera video analytics to sublinear with the number of camera feeds.

2 Tackling the scalability challenge

Consider a canonical multi-camera deployment (Figure 1). For surveillance applications, the objects in the videos constitute the main content of interest, hence the "video semantics". Video analytics systems sift through the video feeds for certain objects. Given the high data volume and the low ratio of *signal* (useful objects) to *noise* (useless background scenes), efficient analytics rests on pruning the search space fast. Existing systems are still far from scaling analytics sublinearly with the number of feeds and frames.

2.1 Lack of adequate video data abstraction

Video data are gigantic but highly redundant. The key enabler of a scalable multi-camera video analytics system is to fully understand and leverage this inherent redundancy. However, there is insufficient understanding and characterization of the redundancy in the current multi-camera video surveillance landscape, impeding further optimizations.

Types of redundancy within videos. Figure 2 shows 4 consecutive images captured by the same camera. We can always see the same car bounded by the red box across all these frames. This is the type of *frame-level similarity* already leveraged extensively in video coding and analysis [37, 45]. Figure 3 shows snapshots from two live video feeds captured by surveillance cameras mounted at the same intersection in Jackson Hole, WY [5]. We can see significant overlap between the views of these two cameras. This is a form of *camera-level spatial-temporal similarity* [42, 43] due to the physical camera placement. Figure 4 shows two images captured by video cameras installed in different parking lots from the VIRAT dataset [58]. Unlike that in Figure 3, spatially these two cameras are not necessarily near each other. However, since both can be viewed as “parking lot" cameras, the video feeds captured by these two cameras are somewhat similar to each other. For example, both of them might contain cars and people stepping out of their cars. We refer to this as *stream-level semantic similarity*, where a *stream* is a contiguous block of frames within a camera feed. We make a distinction between “camera-level” and “stream-level” since a long camera feed may be divided into multiple streams, each featuring different objects (hence “semantics”). As a simple example, consider the video feed from a camera mounted near a railway track. The frames capturing an approaching train are semantically different from when no train is present.

Existing views and limitations. Current data organization in video query systems can be seen as an extension of image retrieval systems, built on top of the *frame-level similarity*. All video data are treated as a giant collection of video frames. However, this requires processing every frame within the video collection at ingestion or query time, which incurs incredibly high computation overhead. While frame sampling or adding camera-level constraints [37, 43, 88] can reduce the number of frames processed, the number of frames extracted from the streaming video data is still huge.

A new abstraction. Clearly, there is a mismatch between existing data abstractions and the fundamental video data characteristics. This necessitates a mechanism to aggregate video at a sub-feed level to both accurately preserve the frame-level video content and

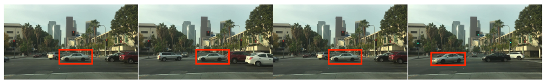


Figure 2: Images captured by the same camera

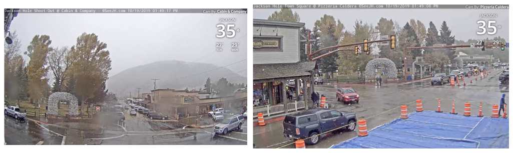


Figure 3: Images captured at the same intersection

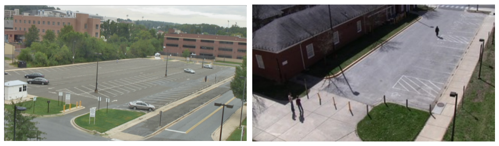


Figure 4: Images captured in different parking lots

minimize the number of data objects to handle. We therefore argue for a level of abstraction between the feed level and the frame level.

We propose a new data abstraction, semantic video stream (SVS), that can represent the content of a subset of a feed. Video frames from a single camera can be divided into several video streams based on their content difference, likely due to scene changes. SVSs aggregate frames to drastically prune the search space. Besides, SVS-based data organization lends to large-scale inter-feed processing that is currently difficult. Examples are specialized neural network training and aggressive video archiving. A semantic understanding of video streams makes it possible to train one specialized neural network per-cluster of SVSs. For the latter, instead of observing the access pattern of individual frames, we can operate at the level of an SVS or a cluster of them.

2.2 Multi-faceted policy considerations

The rise of edge data source. As data collection is increasingly at the edge of the network [40], the data volume at the edge is growing exponentially. Further, given how pervasive surveillance cameras are deployed, the captured data tends to raise privacy concerns [24]. Feeds from different cameras may need to be isolated depending on who owns the camera and manages the captured videos. It is desirable to minimize the number of raw frames sent both for bandwidth and privacy considerations.

Our approach. We organize video data using a hierarchical index, with inter-camera and intra-camera components. The hierarchy can be naturally mapped to different processing locations. Instead of sending all raw data to the cloud, we can send only the representative video streams, and only raw frames containing objects of interest can leave the edge processing point near a camera. Besides largely reducing network overhead in the backbone network, this offers privacy support by preserving the data boundary and a summary view across cameras to the cloud to speed up subsequent query processing.

2.3 Intertwined management and analytics

Existing video stores treat video data the same as any data, and do not provide any support to characterize the redundancy. This shifts the burden to the query systems. Recent video analytics systems implement built-in custom strategies to reduce redundant data processing and scale analytics [37, 43, 88].

Scaling implementation efforts. Building an effective video index requires notable domain knowledge of video processing [42]. Apart from the merit of the processing strategies, including an index within each system inevitably requires repeated implementation of common processing steps, which itself is inefficient. Further, this indicates a lack of separation between basic data organization and actual analytics that makes it harder to scale in other ways.

Towards a data organization layer. Instead, video data analysis and organization should ideally be a separate layer overlaid on a video store and support common types of video queries. For video surveillance, two common types revolve around *specific object identification* (*direct queries*) and *correlating camera feeds* (*clustering queries*). The former takes the format of *find video streams that contain object X in N streams.* This forms the basis for various applications like *theft detection* [26], *incident investigation* [29] and even *atmospheric administration* [10]. The latter is in the form of *find all video streams that are semantically similar to a video stream Y*, which lends to effective specialized network training and video archiving. Additional qualifiers over a subset of camera or time range can be easily supported.

2.4 Introducing *Video-zilla*

So far, we have motivated the need for a new abstraction, a hierarchical index exposing boundaries between cameras, and a system to separate and interface with the video store and the query engine. *Video-zilla* is designed to address these. We outline the challenges and solutions below.

Deriving semantic video streams. Previously, we gave a qualitative illustration of two semantically similar video streams. To be integrated into an analytic system, we need a suitable metric, i.e., a quantifiable definition of "semantics", to delineate SVSs to provide an effective abstraction. On that basis, we need to measure the "similarity" between SVSs for further aggregation. Section 3 describes representing a semantic video stream as a feature map and computing the similarity score between two streams based on a novel metric, *object mover's distance*. We also introduce an approximation method to reduce the computation overhead.

SVS clustering for index construction. We need an index that is precise and concise, capturing the video content comprehensively as well as the level of redundancy. This requires low-latency and effective clustering of all SVSs incrementally as new SVSs arrive in a streaming fashion (Section 4).

Expressive interfacing. As a generic indexing layer between a video store and query applications, *Video-zilla* should provide clean and expressive interfaces to hide the complexity of the underlying data organization but capture the video content. Sections 5 and 6 describe the architecture of *Video-zilla*, the functionality it provides, and the APIs exposed.

3 Semantic video streams (SVSs)

In this section, we define an SVS and a quantifiable notion of "semantics" (Section 3.1). We then propose a novel metric, *object mover's distance (OMD)*, to compare the semantics among different video streams, as well as an approximate algorithm (Section 3.2) to speed up OMD computation. Finally, we discuss how to represent several semantically similar SVSs using a representative SVS (Section 3.3), a primitive for subsequent indexing operations.

3.1 Defining Semantic Video Streams

We define a *video stream* as a contiguous block of frames within a camera feed. The "semantics" of a video stream describes the content of these frames. For object-identification based surveillance applications, the content of each frame can be characterized by the objects captured in that frame. Therefore, we characterize the *semantics* of a video stream with all the objects within that stream, with per-object feature vectors. A *semantic video stream (SVS)* then is the collection of these feature vectors (i.e., the *feature map*).

Note that an SVS only captures the object distribution that may correspond to an event or scene change, but cannot and does not aim to explicitly identify the event. For example, a train-station camera mainly captures two types of scenes interleaved with each other: *train passing*, and *empty tracks*. These will be mapped to different SVSs, some featuring trains. An object distribution of "85% train and 15% people" may correspond to "train passing". The analytics application using these SVSs may *infer* the corresponding events if provided with additional contextual information about the video. Further, an SVS cannot track object motion, which requires tracking more frame-level information than the object distribution.

Video frame clipping. In preparation for deriving SVSs, we first clip objects that is contained in each frame using an object detection mechanism. The notion and derivation of SVS are orthogonal to per-frame object detection, so we can use either lightweight ones such as the YOLO series [65] or more sophisticated approaches [53, 73, 89] to balance the computation footprint and detection accuracy trade-offs. In our evaluation we use YOLO. Each object is represented by its four-point 2-D coordinate (top, left, bottom, right) in the original frame.

Object feature extraction. To characterize an SVS, the first key step is to obtain the feature vector per object. We can lean on state-of-the-art image classification tools such as convolutional neural networks (CNNs) [35, 51, 74]. The output of a CNN is the probabilities of all object classes, and the class with the highest probability is the classification result. The penultimate layer's output of a CNN proves to be representative features of an input image [48]. The features form a real-valued vector, whose length ranges from 512 to 4096 in the latest CNNs [35, 76]. Therefore, running a CNN until the penultimate layer on any image containing an object extracts the feature vector for that object. For a video frame with multiple objects, we can clip the frame and obtain the feature vectors for all objects within the frame.

3.2 Distance between SVSs

The key to our data indexing system is to cluster semantically similar SVSs to reduce the search space for queries. Therefore, we need a mechanism to quantify the similarity between SVSs.

SVS distance. Since the SVS semantics refers to the objects captured, "semantic similarity" between two SVSs manifests as similar objects and their distributions in different SVSs. An SVS distance metric should then reflect how far object distributions deviate between SVSs, i.e., the *travel cost* between objects in different SVSs. As objects are represented by feature vectors, one natural measure is the Euclidean distance in the feature vector space, defined as

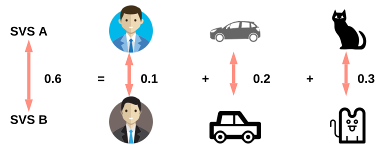


Figure 5: Illustration of Object mover's distance

$d(i,j) = ||\boldsymbol{x}_i - \boldsymbol{x}_j||_2$, where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are the feature vectors of objects i and j.

The "travel cost" between two objects is a natural building block for the distance between two SVSs. Let $\boldsymbol{d}$ and $\boldsymbol{d'}$ be two feature maps, containing n and n' feature vectors, respectively. We allow each object i in $\boldsymbol{d}$ to be either partially or fully transformed into any object in $\boldsymbol{d'}$. Let $T \in \mathbb{R}^{n \times n}$ be a cost matrix where $T_{ij} \geq 0$ denotes the unit travel cost from object $i \in \boldsymbol{d}$ to object $j \in \boldsymbol{d'}$. To transform $\boldsymbol{d}$ entirely into $\boldsymbol{d'}$, we need to ensure that the overall outgoing flow from object i equals d_i, i.e., $\sum_j T_{ij} = d_i$. Further, the amount of incoming flow to object j should match d'_j, i.e., $\sum_i T_{ij} = d'_j$. This way, we define the distance between two SVSs as the minimum cumulative object travel cost required to move all objects in $\boldsymbol{d}$ to $\boldsymbol{d'}$, i.e., $\sum_{i,j} T_{ij} d(i,j)$.

Object Mover's Distance. Calculating the aforementioned minimum cumulative object travel cost can be formalized as:

$$\min_{\boldsymbol{T} \geq 0} \sum_{i,j=1}^{n} \boldsymbol{T}_{ij} d(i,j) \text{ where} : \sum_{j=1}^{n'} \boldsymbol{T}_{ij} = \frac{1}{n}, \forall i \in 1, \ldots, n \qquad \sum_{i=1}^{n} \boldsymbol{T}_{ij} = \frac{1}{n'}, \forall j \in 1, \ldots, n' \tag{1}$$

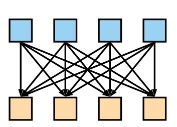
$d_i = 1/n$ and $d_j = 1/n'$ mean that we treat every feature vector within an SVS equally by giving them the same weight. The weights in each SVS sum up to 1. Equation 1 aims to minimize the weighted cost of transferring ("mapping") all feature vectors in $\boldsymbol{d}$ to $\boldsymbol{d'}$. A vector i in $\boldsymbol{d}$ could be mapped to several vectors in $\boldsymbol{d'}$, with its weight d_i split over the target vectors' in $\boldsymbol{d'}$. Therefore, the mapping is one-to-many, not one-to-one, as shown in Figure 6a.

This is a special case of the well-studied transportation problem, Earth Mover's Distance [72], with specialized solvers [52, 60]. To highlight this connection, we use the term *object mover's distance* (OMD). OMD is a metric since $d_{i,j}$ is a metric [71]. Figure 5 illustrates the OMD metric for two SVSs, A and B. Each arrow represents the correspondence (or *flow*) between two objects, their Euclidean distance shown next to the arrow. The distance between two SVSs reflects a cumulative travel cost across the objects.

Fast OMD computation. Calculating an accurate OMD between two SVSs has $O(n^3 \log n)$ time complexity, where n is the number of features within a video stream. It is thus impractical to compute the OMD between large video streams.

Fortunately, there are several fast specialized approximate computation methods [52, 59, 60]. We adopt the thresholded ground distance method [60]. Instead of comparing all pair-wise object moving costs, we set a distance threshold t. Any pair-wise distance between feature vectors larger than t will be capped to t. We essentially merge all such pair-wise relationships to a single group.

Figure 6 illustrates the transformation process from the original OMD computation to the thresholded-OMD computation. The boxes are objects and the colors indicate different video streams. The red circle is the new transshipment vertex that serves as an


(a) Original OMD

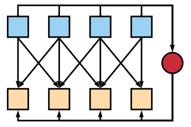

(b) Thresholded OMD

Figure 6: Illustration of the thresholded OMD

intermediate vertex, before a feature vector goes to its original destination. Any incoming edge cost is the threshold t and any outgoing edge cost is 0. The drastic reduction of the number of edges is the main reason for computation time reduction when using FastOMD.

3.3 Representative construction

To efficiently cluster SVSs, we also need a way to construct a representative SVS for an SVS cluster. A new SVS can then be compared with per-cluster representatives, rather than with all SVSs in that cluster. Section 4 explains how to identify a suitable existing cluster for a new SVS to join.

k-clustering representative construction. Recall that objects that are visually similar have similar feature vectors in the Euclidean space. We perform k-means to derive k clusters of feature vectors, each has a centroid feature vector. Weighted by the size of each cluster, these centroids‘ vectors then form a representative SVS.

We adopt the silhouette method to determine the value of k [68]. The silhouette value is a measure of how similar an object is to its own cluster compared to other clusters. Formally, for data point $i \in C_i$, $s(i) = \frac{b(i)-a(i)}{max\{a(i),b(i)\}}$, where $a(i) = \frac{1}{|C_i|-1}\sum_{j\in C_i, i\neq j} d(i,j)$ and $b(i) = \min_{k\neq i}\sum_{j\in C_k} d(i,j)$. A high silhouette value indicates a well-formed cluster.

Defining query hit. To facilitate direct object identification queries, we record the boundary for each weighted center. The boundary is defined by the distances between the farthest data points in all directions and the cluster center. We obtain a query *hit* when the queried feature lies within the boundary of a weighted center.

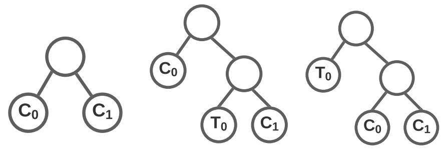
4 Clustering SVSs

Given a set of SVSs, index construction for fast query boils down to clustering the SVSs based on the similarity of their semantic content. In this section, we first explain an existing incremental clustering algorithm that works for both the Euclidean and OMD metric space, explaining the basic data structure and related operations in Section 4.1. Section 4.2 then maps these basic operations to those needed for an SVS index. Finally, we propose a novel pruning-based approximation technique to largely reduce the computation overhead incurred by OMD computation in Section 4.3, which is our unique system contribution when dealing with clustering in the OMD space.

4.1 Basic data structure and operations

SVS organization. Taking a leaf out of previous work [47], we organize SVSs with a tree. Each leaf node in this tree represents a unique SVS. The root node represents all SVSs stored in the index. Each internal node is the root of a subtree and represents all SVSs at the leaves in this subtree. Each children node of the same parent node contains a subset of SVSs of that in parent node. As clustering SVSs can be seen as partitioning a set of SVSs into multiple subsets, the SVS tree structure encodes multiple ways of clustering SVSs.

Evaluating cluster tree using dendrogram purity. The quality of a cluster tree is less obvious compared to a “flat” clustering approach, such as k-means. We adopt a holistic measure, known



(a) Initially (b) C_0 masks T_0 (c) Unmask T_0

Figure 7: Masking and unmasking

as *dendrogram purity* [36]. In words, the dendrogram purity of a cluster tree with respect to a ground truth clustering C^* is the expectation of the random process computed in the following steps: 1) sample two SVSs, s_i and s_j uniformly from the same cluster C_x in the ground truth; 2) compute their least common ancestor p in the cluster tree T; and 3) compute the fraction of leaves of p that also belong to C_x. With this definition, the dendrogram purity is a real number between 0 and 1. A "perfect" cluster tree has dendrogram purity 1, in which all leaves of an internal node belong to the same ground-truth cluster.

Greedy incremental clustering. For live video feeds, new SVSs arrive continuously, hence it is natural to follow a greedy incremental insertion strategy. For each incoming SVS x, the algorithm will iterate through all the leaf nodes in the cluster tree to find the nearest neighbor to x, a leaf node s. A `Split` operation is performed on s that first disconnects s from its parent, creates a new leaf node s' to store x, and finally creates a new internal node, whose parent is s's former parent and whose children are s and s'.

Masking and unmasking. However, this greedy insertion strategy cannot reach an optimal dendrogram purity because of the *masking effect*. Figure 7 illustrates the case when the current leaves in a tree belong to a "car-dominant" cluster, denoted C_0 and C_1. If an incoming SVS T_0 belongs to a "train-dominant" cluster, according to the greedy incremental tree construction heuristics, it will be first inserted in the tree next to C_1. We say T_0 is masked by C_0, since C_0 and C_1 are clearly more similar to each other and should be sibling nodes. Masking effect essentially means putting a SVS is being covered by a supposed representative that does not actually represent it well. The root cause of masking effect is that the random incoming order of SVSs cannot guarantee siblings in a cluster tree are actually the closest (like T_0 and C_1 in Figure 7b).

Formally, A node v with sibling v' and aunt a (i.e., the sibling of a parent node) in a tree is masked if there exists a point $x \in lvs(v)$ such that, $\max_{y\in lvs(v')} OMD(x,y) > \min_{z\in lvs(a)} OMD(x,z)$. $lvs(z)$ is the set of leaves for any internal node z. [47] proves that a cluster tree without masked nodes has dendrogram purity 1.

Inspired by self-balancing trees like red-black trees [28], we employ the following masking triggered rotations to mitigate the dendrogram purity loss. When masking is detected, a `Rotate` operation swaps the position of s with its aunt in the cluster tree. After the rotation, the algorithm checks if s's new sibling is masked and recursively applies rotations until no masking is detected or we reach the root (Algorithm 1). The intuition behind the rotations is to move the "masked" node up towards the root by swapping this node with its aunt recursively, until masking no longer manifests.

4.2 Building an SVS index

SVS insertion. The combination of greedy incremental insertion and masking-triggered rotation provides a baseline algorithm to

Algorithm 1 Rotate(v, conditionDetect)

```
(ShouldRotate, ShouldStop) = conditionDetect($v$)
if ShouldRotate then
    $Tree$.Rotate($v$, conditionDetect)
end if
if ShouldStop then
    Output: $Tree$
else
    Output: $Tree$.Rotate($v$.Parent(), conditionDetect)
end if
```

insert an incoming SVS (Algorithm 2). This is referred to as purity enhancing rotations for cluster hierarchies (PERCH [47]), building a cluster tree with optimal dendrogram purity. Further, the algorithm also performs a balancing-triggered rotation. This is an optimization we will discuss in the following subsection, together with a pruning-based nearest neighbor search optimization.

SVS clustering. Given the SVS tree, we derive the actual clusters using the following heuristics. We maintain a list of tree nodes to represent different clusters. Initially, only the root node of a tree is in this list. While the number of nodes in the list (i.e., the number of clusters) is less than k (determined by the silhouette method mentioned in Section 3.3), we iteratively remove the tree node with the smallest *cost* in the list, replacing it with its two children nodes added to the list. The clustering process terminates once there are exactly k tree nodes in the list. The *cost* of a tree node v is defined as the maximum distance among the $lvs(v)$.

Search operations. Our index supports two basic search primitives: *feature search* and *SVS search*, corresponding to the direct identification query and clustering query. A *feature search* aims at finding a set of SVSs that might contain feature vectors that are similar to the target feature vector. We use the decision-boundary-based query (Section 3.3), first to identify candidate clusters of SVSs, and then searching all SVSs in candidate clusters to find the SVSs that actually meet the requirement. An *SVS search* aims at finding the SVS that is closest to the target SVS. This is supported by the nearest neighbor search function in the insertion algorithm.

Smoothness of the OMD space. The latent space generated by SVSs may not be smooth, especially when an SVS consists of high-dimensional feature vectors. Due to the *curse of dimensionality* [23, 79], we cannot avoid this issue completely. This "discontinuity" causes some SVSs to be inserted into the wrong clusters, and a subsequent query to *Video-zilla* may either miss an SVS (false negative) or examine an SVS unnecessarily (false positive).

Given the nature of video scenes, we cannot derive close-form expressions for this discontinuity. To quantify its impact empirically, we evaluate the false-positive (FPR) and false-negative rates (FNR) of queries with and without *Video-zilla* in Section 7.4. Further, we also introduce a performance adaptation and bailout mechanism in Section 5.3.

4.3 Nearest neighbor search optimization

One sub-procedure of the algorithm in 4.1, finding nearest neighbors in the cluster tree, can make its naive implementation slow. In this section, we introduce pruning-based nearest neighbor search with the help of an easy-to-compute lower bound of OMD. In addition, we perform a balancing based rotation operation as further optimization.

Algorithm 2 Insert(x_i, $Tree$)

```
$nbrs$ = NearestNeighbor($x_i$)
$leaf$ = Split($nbrs$)
for $a$ in Ancestors($l$) do
    $a$.AddPt($x_i$)
end for
$Tree$ = $Tree$.Rotate($leaf$.Sibling(), CheckMasked)
$Tree$ = $Tree$.Rotate($leaf$.Sibling(), CheckBalanced)
Output: $Tree$
```

Pruning-based search. We propose a novel pruning technique by introducing a cheap lower bound of the OMD that allows us to prune away the majority of SVSs in the cluster tree without ever computing the exact OMD distances.

Following [71], it is straightforward to show that the distance among the centroids of feature vectors within two SVSs is a lower bound for the OMD between them. We refer to this lower bound as the object centroid distance (OCD) as each SVS is represented by the weighted average vector over all feature vectors contained in it. Compared to OMD, OCD is easy to maintain and update in $O(N)$ time, where N is the feature vector dimension.

We use OCD to drastically reduce the amount of OMD distance computation when performing the nearest neighbor search. We first sort all SVSs within the cluster tree in ascending order of their OCD distance to the target SVS. Starting from the head of the sorted list, we repeatedly compute the OMD between the target SVS and the SVS that is at the head of the sorted list. The distance of this SVS will be updated and the list will be sorted based on the updated distance. As the OCD is a lower bound of OMD, we maintain a set of explored SVSs whose OMD has been computed. The first SVS we visited that has already been explored is guaranteed to be the nearest neighbor of the target SVS.

Although pruning-assisted process can largely reduce the computation overhead of nearest neighbor search, it cannot reduce the OMD computation when checking the masking condition or updating the *cost* for each internal node. As the masking condition is checked in a bottom-up manner and the check terminates when no further masking effect is detected, only a small subset of the SVSs within the index are involved. For the *cost* update for each internal node in the tree, we adopt a bottom-up approximation heuristic. The algorithm updates the *cost* of the nodes along the path from the leaf to the root, terminating once the *cost* of a node within the path does not change.

Balancing based rotations. Our pruning method can be largely affected by the depth of the tree. While our rotation algorithm guarantees optimal dendrogram purity, it does not provide any guarantees on the depth of the generated cluster tree. The balance of a cluster tree T is the average local balance of all nodes in T, where the local balance of a node v with children v_l, v_r, is $bal(v) = \frac{min\{||lvs(v_l)||,||lvs(v_r)||\}}{max\{||lvs(v_l)||,||lvs(v_r)||\}}$. Similar to the masking condition, we will rotate the current node with its aunt in the tree if we find that it will improve the balance of the tree without causing masking.

5 *Video-zilla* system

Based on the clustering mechanisms described previously, *Video-zilla* (Figure 1) builds a hierarchical index with two components: 1) an *intra*-camera index per camera feed to index the video streams captured by the same camera; 2) an *inter*-camera index across all cameras to index the representative semantic video streams constructed by all intra-camera indices. The *inter-camera* index resides somewhere centrally, where the queries are issued, while the *intra-camera* indices could stay at edge servers to filter raw data and ensure most of them remain local. Each camera sends raw video data to the nearest edge server, via wired or wireless connections. The hierarchical index maintains the boundary between cameras, which allows the components to map to different processing locations when appropriate.

Video-zilla is designed as a layer between the video store and applications. Consider Microsoft's Project Rocket [9], for example. *Video-zilla* can be interposed between the light and heavy DNN detectors, using the feature vectors extracted by the light DNN detectors to build the index, and the heavy DNN detectors to refine the final query results.

5.1 Hierarchical index generation

An incoming image frame will first pass a key frame selection module, which adaptively filters video frames based on the computation capacity of the edge server. The selected key frames will then pass several feature extraction modules to compute application-specific feature vectors. Our automatic video segmentation module will then segment a stream of feature vectors generated by the same camera into separate SVSs. These SVSs will be first inserted into the corresponding intra-camera index. Finally, triggered by the representative SVS update in an intra-camera index, the per-model inter-camera index will be updated. Following this process, our hierarchical index will be constructed incrementally.

Automatic video segmentation. Given a stream of feature vectors, the first key step is to derive an SVS from them. This can be especially helpful for scenarios with frequent scene changes, e.g., as a drone flies above different terrains. We propose a greedy segmentation heuristic. The general idea is to track novel features in consecutive frames and then segment the video when a set of features seems to drift away from the previous SVS.

Initialization. To bootstrap the system, the initial portion of the streaming video over a set length of time t_{max} is extracted as the first SVS. This t_{max} also serves as the maximum length of an SVS. The choice of t_{max} will affect the query result granularity. In practice, we leave the choice of t_{max} to the application developer. In our evaluation, we set t_{max} as 15 minutes.

Tracking novel features. After initialization, we use the representative SVS of the cluster that the last segmented SVS belongs to as the reference SVS. We maintain a current feature buffer that contains all incoming feature vectors after the last video segmentation. An incoming feature vector that lies outside the decision boundary (Section 3.3) of the representative will be labeled as a novel feature. We track all novel feature vectors using a novelty feature buffer.

Video segmentation. Whenever a novel feature vector is added to the novel feature buffer, we cluster feature vectors in the buffer using k-means and calculate the average distance between the members and the corresponding cluster center. We compare this average distance d_n in the novelty buffer with d_r in the representative. Video segmentation (Algorithm 3) is triggered when $d_n \leq d_r$. At the same time, we also record the hit pattern of the weighted clustering centers in the reference SVS. if one of these centers has

Algorithm 3 Video segmentation

```
% A novel incoming feature vector $v_{novel}$
NovelFeatureBuffer.add($v_{novel}$)
CurrentFeatureBuffer.add($v_{novel}$)
$d_n$ = NovelFeatureBuffer.calAvgDist()
$d_r$ = SVSTree.avgRepDist()
$t_{hit}$ = SVSTree.maxLastHitTime()
if $d_n \leq d_r$ or $t_{hit} > t_{split}$ then
    $SVS_{new}$ = Features(CurrentFeatureBuffer)
    insert($SVS_{new}$, $SVSTree$)
end if
```

not been hit by any incoming features after time t_{split}, the video needs splitting. t_{split} is set to be $\frac{1}{10}t_{max}$ empirically. To segment the video, the current feature buffer is divided at the point where the first novelty feature or the last hit feature arrives.

Take a train-station surveillance camera as an example. It should mainly see trains and people (with luggage) or a largely empty platform. The video feed captured by this camera will be divided into chunks using our automatic video segmentation mechanism, correlating with train arrival and departure.

Hierarchical index update. The newly generated SVS is inserted into the corresponding intra-camera index. One or more representative SVSs in this intra-camera index will be updated. The updated representative SVSs will then replace the outdated versions in the inter-camera index. This completes one update round of our intra- and inter-camera hierarchical indices.

Adaptive key frame selection. It is neither computationally feasible nor efficient to process every single video frame captured by a video camera. This module determines which video frames should have objects clipped and passed to the feature extraction module. The ingestion computation cost is determined by both the frame rate and the inter-frame deviation threshold. The latter one uses the deviation between two consecutive images as the metric. When the deviation exceeds a threshold t, we consider the incoming image as a key frame. Many combinations of these two factors are plausible, and we adopt a best-effort heuristic to avoid queuing. We monitor the input frame queue at the feature extraction module. Once a queue starts building up, we will downgrade it to a more lightweight configuration. Conversely, we will upgrade it to a more heavyweight configuration.

Customizable feature extraction. As different applications may prefer specific feature extraction techniques, we make this module customizable. Each application can register their own feature extraction module. We provide several default feature extractors, including VGG16, VGG19, ResNet50, and ResNet101 [8], all trained on the COCO dataset [3].

5.2 Query processing

For direct object identification, the query specification includes an image containing the object of interest, together with the optional time range and camera ID constraints as the metadata. The candidate representatives SVSs will be first identified in the inter-camera index. Then the query will be dispatched to all the intra-camera indices containing the candidate representatives, to search for the actual SVSs that contain the queried object.

A clustering query takes as input a feature map that characterizes an SVS, as well as the optional time range and camera ID constraints.

For this query, *Video-zilla* returns all similar SVSs to the input SVS. *Video-zilla* will check the inter-camera index to find the cluster C containing the most similar SVS to the input SVS. Each intra-camera index will return all SVSs that belong to the cluster represented by a representative SVS in C.

5.3 Performance monitoring and bailout

Performance monitoring. Due to the errors induced by feature extraction, index building and SVS candidate selection, querying our hierarchical index cannot guarantee 100% accuracy. Thus, *Video-zilla* monitors the error rate of query results and adjusts the index setting to satisfy the user-defined error preference. Periodically, in addition to querying the hierarchical index, *Video-zilla* runs the same query on all the video frames in the background. This query result serves as the ground-truth. This will inevitably take a long time, so *Video-zilla* only performs this operation every 50 queries.

If the current query F1 scores do not meet the user preference, *Video-zilla* can adjust the following parameters one at a time: i) increasing the number of clusters within the inter- and intra-indices; ii) decreasing the OMD computation threshold to derive a more accurate OMD value; and iii) downgrading to a flat SVS index, i.e., without distinguishing between the intra- or inter-camera indices.

Bailout. If applying all three parameter adjustments still cannot meet the user error preference, a bailout mechanism will be triggered, where *Video-zilla* will downgrade to a frame-level index to search through video frames across all cameras. Meanwhile, *Video-zilla* periodically runs a query on the hierarchical index to determine when to switch back to the hierarchical index. *Video-zilla* performs this operation every 10 queries.

5.4 Discussion

Privacy considerations. Orthogonal to common privacy protection approaches such as data encryption and image anonymization (via pixelation or blurring), *Video-zilla* introduces some amount of anonymization through feature aggregation. Given our hierarchical index design, only the representative SVSs in each intra-camera index will be sent to a centralized operation point to construct an inter-camera index. An analytics user can only access the few frames returned as the query results and not the raw videos at large.

Security concerns. IoT cameras have been vulnerable to security attacks or exploited in DDoS attacks (such as Mirai [67]). While this work does not explicitly address security issues, *Video-zilla* may help by acting as an intermediary, which facilitates limiting direct access to the cameras.

Per-model indexing. *Video-zilla* generates an index per DNN model. However, the index generation process is the same, regardless of the feature extractor, and hence different applications can run on a common indexing layer.

Camera ID and time range filtering. We can filter the camera ID at the *inter*-camera level when identifying the *intra*-camera indices to dispatch the queries to. The time range can be applied in each intra-camera index, by comparing against the video timestamps.

6 Implementation

We implement *Video-zilla* using the Akka toolkit [1]. Following the actor model in Akka, *Video-zilla* comprises of five actors, key frame selection, feature extraction, intra-camera index, inter-camera index and query history cache. Data exchange between modules follows asynchronous message passing. There is also a query actor per type of queries.

We use the OpenCV library [21] to implement key frame selection, including video I/O, frame rate control and image deviation computation. Keras 2.3.1 [7] is used to train the default feature extractors, and the massive online analysis library [19] for representative construction. The fastOMD implementation adapts the fastEMD library [6]. Our incremental clustering algorithm extends the codebase in [47] to our OMD metric space. Each intra-/inter-camera index is stored as a tree structure.

Our feature extractors are trained using Microsoft COCO, and this code is written in Python. Other *Video-zilla* modules are written in Java. The implementation totals about 4K lines of code, including 3K in Java and 400 in Python for the indexing, and around 500-600 lines of Java code for the query stubs. The codebase size of *Video-zilla* suggests that existing analytics systems can be simplified substantially by issuing queries to a generic index in place of performing custom video content analysis per application.

Setup and configuration APIs. The analytics application can add or remove a feed with `cameraStart(cameraID, historyDataTimeRange, appID)` and `cameraTerminate(cameraID, appID)`. We make the feature extraction module pluggable and expose an API to applications to customize feature extractors. `setFeatureExtractors(Model, appID)` allows an application to specify a custom feature extractor to characterize the video semantics.

Query APIs. *Video-zilla* supports two most common queries, `directQuery(objectImg, appID)` for direct object identification, and `clusteringQuery(targetSVS, appID)` to organize SVSs. A direct query takes as argument the image of the object of interest, while a clustering query takes as input the feature map of an SVS. We further provide a utility API, `getMetaData(SVS)`, to return information such as the start and end timestamps of this SVS, the camera ID that captured this SVS, and the access time of this SVS.

Customizable APIs: Video archiving as a case study. Developers can also implement custom APIs by composing the built-in APIs. For example, to build a video archiving service, we need an API `isArchived(targetSVS, appID)`, which is a variation of the clustering query. Instead of returning a list of semantically similar SVSs, it returns the average access frequencies of all these SVSs. Code snippet 6 shows how to implement this by composing `clusteringQuery` and `getMetaData`.

```
isArchived(targetSVS, appID):
    SVS[] res = clusteringQuery(targetSVS, appID);
    sumFreq = 0;
    for (SVS svs : res):
        sumFreq += getMetaData(SVS).accessFreq;
    return sumFreq / res.length;
```

7 Evaluation

Hardware setting. We use two Linux servers to host the hierarchical index, both with 8-core 2.1 GHz Intel Xeon CPUs, one with a NVIDIA RTX 2080Ti GPU, and the other with an NVIDIA RTX 2070 GPU. We run an inter-camera index on the former and intra-camera indices on the latter. In cases we need to run multiple intra-camera indices, this process is done in a serial fashion on a single machine. This way we avoid any artifacts that may arise from a distributed setup. *Video-zilla* is orthogonal to improving

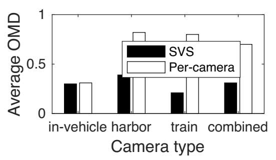
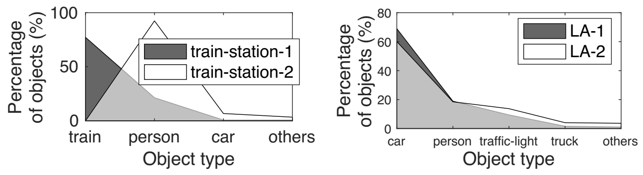


Figure 8: OMD comparison.



(a) Train-station camera **(b) In-vehicle camera in LA**

Figure 9: Object distributions from the same feed.

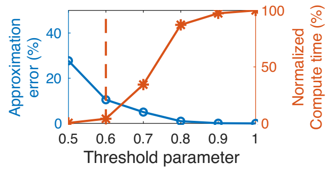


Figure 10: The impact of threshold on FastOMD.

processing efficiency with distributed computing, and can leverage existing works [16, 39, 88] on that.

Datasets. For the microbenchmarks (Sections 7.1, 7.2, 7.3), we synthesize a dataset from 1000 SVSs. Each contains 500 1024-dimension feature vectors, and these vectors follow a multivariate normal distribution. We assume there are 10 different types of feature vector distributions in total, each containing 100 SVSs.

For the end-to-end case studies (Sections 7.4, 7.5, 7.6), we assemble publicly available real-world datasets to emulate a real-world multi-camera surveillance system for public transportation, like that in Chicago [13]. The total length of these videos is 30 hours, including three types of feeds from both stationary and moving cameras: i) 40 road-view captured by in-vehicle cameras: 20 of them capture the downtown areas [4] of New York City, London, Chicago and Los Angeles, 5 per city; the other 20 capture highways across the U.S. [20]; ii) 2 train-station video livestreams from Youtube [62, 63]; and iii) 2 harbor video feeds from Youtube [78, 81]. We intentionally set the number of cameras in train-stations and harbors to be smaller than that of in-vehicle cameras, which matches the expected ratio between these feeds in practice.

7.1 Comparison of video characterization

SVS vs. Camera-level. We compare the SVSs from the same camera feed with those belonging to the same semantic cluster (which may or may not be from the same camera). For both sets of SVSs, we compute the average OMD distance between the SVSs pairwise. Besides the first three real-world scenarios, we derive a combined case by concatenating a downtown road-view video with a highway-to-national park road-view feed. This emulates a car driving from a downtown area to a highway, to increase the variability of content.

We use four types of feeds, including 10 in-city and 10 on-highway camera feeds as the in-vehicle case, 2 "harbor" feeds, 2 "train-station" feeds, and 10 camera feeds as the last combined case. The average length of SVSs extracted by *Video-zilla* is around 10 minutes worth of video, 600-700 frames.

Figure 8 shows that the average OMDs of in-vehicle cameras are similar to one other, as the feeds we captured are relatively short and the frame content within each feed is fairly homogeneous. Empirically, the OMD among the SVSs within the same *cluster* is around 0.3 and 0.35. However, there is a distinct difference between SVSs in the other three cases. A lower average OMD for *Video-zilla* means that *Video-zilla* is able to better capture the semantic similarity compared to camera-level video characterization. Further, it shows the OMD calculation works regardless of whether the video frames are from stationary train-station cameras or moving in-vehicle cameras, as the object detection mechanism *Video-zilla* adopts works for both cases. As an example, Figure 9a shows object distributions from two SVSs derived from a train station surveillance camera, and Figure 9b shows the same for an in-vehicle camera in LA. The object distribution over time hardly changes for the LA road-view feed. In contrast, for the train-station feed, the object distribution varies depending on what events are covered. This highlights the descriptiveness of an SVS over a camera-level video feed.

SVS vs. Frame-level. Using frame-level characterization results in many more data objects to handle. Figure 9 suggests that SVSs as data units for queries capture the object distribution, SVS can improve system scalability without sacrificing descriptiveness.

7.2 Effective and efficient SVS derivation

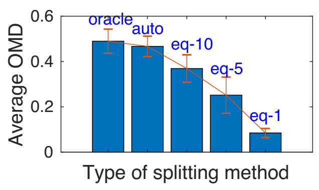
Automatic video segmentation. We select and concatenate 10 feature maps as SVSs from our synthesized dataset. To mimic real settings where SVSs differ in length, the number of composite feature vectors within each SVS is a random number ranging from 250 to 750, and the feature vectors in each SVS follow a distinct multivariate normal distribution.

Our automatic video segmentation technique is compared with two baseline methods: (i) an oracle that can perfectly segment a video feed, which, in our case, means segmenting the video feed exactly into the 10 original SVSs; and (ii) a strawman that simply divides a video stream into equal-length video clips, where the fixed length can be 1, 5 or 10 minutes' worth of video clips.

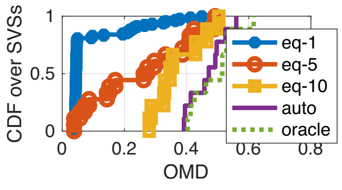
We evaluate the effectiveness of the video segmentation techniques using the average OMD distance between SVSs that appear consecutively in a feed. Higher OMDs mean better segmentation effect. Figure 11a shows the average OMD distance using different segmentation techniques. Our method nearly matches the oracle while outperforming the strawman method. Zooming in, Figure 11b shows the empirical CDF of the OMD between adjacent SVSs. When video clips are segmented evenly, it is unavoidable that some adjacent clips are similar to each other and exhibit low OMDs.

Fast OMD. Figure 10 shows the impact of the FastOMD threshold α on the computation accuracy and time. We randomly select 100 pairs of SVSs from the synthesized dataset, and compute the pairwise OMD distance as α varies from 0.5 to 1. We use the OMD for $\alpha = 1$ as the reference ground truth and calculate the approximation error as the accuracy metric. The computation time is normalized with respect to the computation time needed for $\alpha = 1$.

The approximation error decreases with increasing α at the expense of a higher computation time. Empirically, $\alpha = 0.6$ appears to achieve a balance between the computation accuracy and processing time reduction. For this α value, the average OMD computation time is reduced to less than a second (767 ms on average). Since the average length of the SVSs in the real-world dataset is around 12 mins, this overhead is acceptable.



(a) Average OMD



(b) OMD distribution

Figure 11: OMD between adjacent SVSs

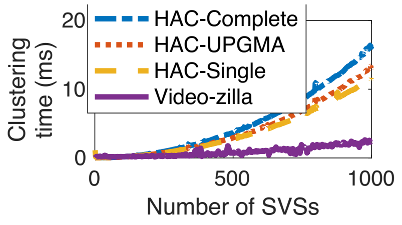


Figure 12: Clustering algorithm comparison.

7.3 Scalable incremental clustering

Pruned nearest neighbor search. Figures 13a and 13b show computation reduction when inserting or querying an SVS given different index sizes, measured by the number of SVSs with the index. For SVS query, the nearest neighbor search is the only trigger for OMD computation, and our pruning approach can reduce the computation by 92%. For SVS insertion, checking for masking and updating the cost of internal nodes incur additional OMD calculations, but pruning still reduces the total OMD computation by 80%.

Incremental SVS clustering. Figure 12 compares Video-zilla's incremental clustering algorithm with the commonly-used hierarchical agglomerative clustering (HAC) algorithms [57] with differing linkage choices. All algorithms achieve similar clustering accuracy. Every incoming SVS triggers a clustering attempt to update the representative SVSs of the index. The overhead of the three HAC algorithms increases quadratically with the size of the index, while that of *Video-zilla* only linearly by avoiding reconstructing the whole tree every time.

Comparison with M-tree. SVS organization leverages a hierarchical incremental clustering tree, which is a well-studied area. We compare our method (PERCH with OMD approximation, or "PERCH-OMD") with M-Tree [25], an efficient incremental indexing method for similarity search in a metric space. Using the synthesized dataset, we perform a 100-nearest-SVS search for 10 randomly selected SVSs. We choose 100 because this is the ground-truth cluster size for each SVS cluster in the synthesized dataset.

In Figure 14, the x-axis represents the maximum number of elements in a single node of an M-Tree, which we refer to as the maximum node size, and the y-axis represents the number of OMD computation needed. For comparison, the dashed line shows the number of OMD computation needed when using PERCH-OMD.

Both PERCH-OMD and M-tree can derive the correct set of 100 nearest SVSs, but M-tree incurs extra OMD computation. This shows that the choice of the maximum node size can heavily influence the number of OMD computation needed. Further, this choice depends on the number of SVSs within a cluster, which varies with the actual video feeds. The main reason is that data structures like M-tree still suffer from the masking effect mentioned in Section 4. Elements which belong to the same SVS cluster can be contained in disjoint subtrees, while some SVSs that should belong to different clusters are included in the same subtree. The extra OMD computation arises from comparing with these extra elements. The original M-tree algorithm may also suffer from potential overlap between different leaf nodes. However, this overlap problem appears to be addressed in most open-source M-tree implementations.

Comparison with approximate nearest neighbor (ANN) search. Since PERCH-OMD performs *precise* nearest neighbor search, we also compare it with its approximate counterparts. Specifically, we compare with a state-of-the-art ANN algorithm [30]. This is one of the most efficient ANN algorithms [2], with a well-documented open-source toolkit [12], including built-in support for the EMD metric space.

As before, we perform a 100-nearest-SVS search for 10 randomly selected SVSs in the synthesized dataset. The average recall for this ANN algorithm is 97.8%, which is even slightly better than reported in the original paper (92.5%). This is due to the difference in the dataset. Still, we can observe accuracy loss when applying ANN. In contrast, PERCH-OMD considers strictly nearest neighbors, which are by nature more accurate than ANN. Efficiency-wise, given we are not dealing with a huge number of SVSs, we do not necessarily need the "most computationally efficient" algorithm.

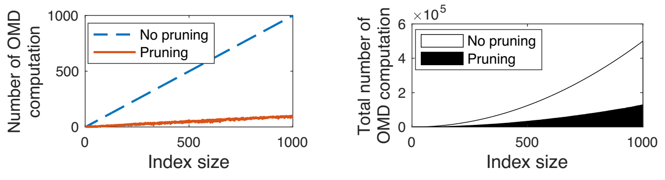
Index building overhead. For an index that contains 1000 SVSs in our synthesized dataset, the overall index size is less than 5 MB. The corresponding video length is around 200 hours, whose data size can be more than 20 GB if the video data are captured in 640 × 480 pixels. The overall index building time is less than 20 minutes, which is negligible compared to the total length of video.

The hierarchical index construction in *Video-zilla* naturally lends to distributed indexing. In contrast, building a single, flat centralized index would send more raw data to a centralized point. We therefore compare the amount of data sent for index construction between these two approaches. Assuming key frame selection and feature extraction done at the edge, we incrementally add 20 camera feeds to the system, each containing 100 SVSs randomly selected from the synthesized dataset. By only sending representative SVSs, adopting a hierarchical index can reduce the network traffic between the cloud and edge servers by a factor of 19.

7.4 Case study: Direct object identification

Direct object identification queries follow the format of *find image frames that contain object X within all streams*. We let X be *fire hydrant*, *boat*, or *train*. This is because the object of interest should be contained in some but not all videos. Searching for *giraffe* or *person* would not make sense, as no videos in our datasets contain giraffes and virtually all contain people. We generate 50 query instances, taken from the real-world videos, per query type.

We compare *Video-zilla* with two state-of-the-art correlation analysis mechanisms in large-scale video analytic systems, per-camera top-k indexing in FOCUS [37] and leveraging spatial-temporal correlation between cameras in Spatula [43]. FOCUS optimizes the processing latency of individual camera feed by building an approximate frame-level index per feed at ingestion time to reduce query time later. Spatula speeds up multi-feed analytics by leveraging the insight that objects found in one camera feed will only



(a) Per SVS query (b) Per SVS insertion

Figure 13: Number of OMD computation.

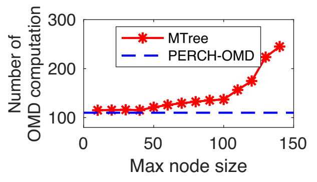


Figure 14: PERCH vs M-tree

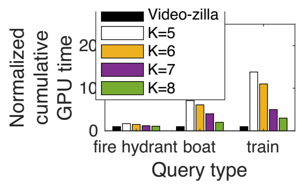


Figure 15: The impact of K

appear on nearby cameras with spatial-temporal correlation with the first camera. Both systems include additional system optimizations such as CNN compression to reduce ingestion overhead and pruning errors via replay search that are orthogonal to *Video-zilla*, so we focus on comparing the query quality and latency for *Video-zilla* based on SVSs vs using the approximate frame-level index in FOCUS or the spatial-temporal correlation between cameras in Spatula. Specifically, we compare the *intra*-camera processing of *Video-zilla* to using the per-camera top-k index and the *inter*-camera processing of *Video-zilla* to a Spatula-like system leveraging spatial-temporal correlation. Unless otherwise stated, all queries in this section achieve at least 95% precision and recall.

***Video-zilla* vs per-camera top-k.** For each camera, we build an approximate top-k index for the incoming image frames. An incoming query will be directly dispatched to the top-k indices of *all* camera feeds, instead of only to the inter-camera index in *Video-zilla*. To ensure the performance is not affected by unrelated factors, we use the same ResNet50 used in *Video-zilla* feature extractor with the additional `softmax` layer to generate the top-k index, and we use the same set of video frames to build indices for both systems. This way, the ingestion overhead of both systems is roughly the same. As in [37], we set $k = 3$ for all top-k indices, i.e., each object in a streaming video will be indexed by the top 3 possible object classes in the corresponding top-k index. We adopt the same Yolo-v2 as the ground-truth CNN in [37].[1] The ground-truth CNN is the main contributor to the query time.

We measure the bottleneck query time (Figure 16), i.e., the time taken for the slowest intra-camera index to return results. This is because the end-to-end query time is bottlenecked by this even when we parallelize query processing of intra-camera indices.

There are two components of the query processing time: one is searching for the right frames, and the other is processing those frames to complete the query. By carefully indexing video data, *Video-zilla* can minimize the search time and thus reduce the total number of frames processed in the latter part, seen in a decrease in the cumulative GPU time, but does not affect the processing time in the slowest intra-camera index. Video-zilla tries to reduce the computation time on unnecessary video data, not to reduce the essential search time.

Figure 17 shows the cumulative GPU time across all intra-camera indices. We do not show the cumulative CPU time here as GPU is typically the resource bottleneck for deep learning inference workloads. While achieving nearly the same query time for all types queries, *Video-zilla* reduces the cumulative GPU time by 14× compared to the top-k indices. Viewed differently, *Video-zilla* can support up to 14× more video queries simultaneously while incurring negligible accuracy loss.

[1]We also tried YOLO-v3 and YOLO-v4, and saw little difference in the object detection accuracy for our datasets. However, it is much faster to run v2.

The root cause is that a frame-level top-k index does not capture enough feed-level information, whereas the hierarchical index of SVSs in *Video-zilla* does. The top-k index incurs unnecessary computation if it mis-classifies objects, whereas *Video-zilla* is more robust. Careful inspection of the top-k indices for each camera feed suggests that a train object can be found even in the top-3 index of a video captured in downtown Manhattan. An actual query therefore causes the system to unnecessarily search through the Manhattan feed. Figure 18 illustrates this mis-classification effect. For the same video segment, *Video-zilla* will identify three object classes, but the top-k index has 4 classes, the fourth being "other". This "other" class is the source of extra, unnecessary computation in a top-k index, as any video frames associated with this class will be examined during an actual query.

We use K to represent the number of classes that can be recognized in a top-k index. Setting a proper K value can help reduce the mis-classification effect. In our case, we set K to 5. Decreasing K from 5 to a lower value will clearly amplify the mis-classification effect in a top-k index. Therefore we increase K from 5 to 6, 7 and 8. Figure 15 shows the impact of the K value on the cumulative GPU time. However, identifying the right K value is non-trivial and requires careful inspection of the specific video feed. The whole point of Video-zilla is to automatically organize data based on its semantics instead. Furthermore, a larger K requires a more complicated recognition model, hence larger processing overhead at ingestion time.

***Video-zilla* vs spatial-temporal correlation.** The main purpose is to show the benefits and caveats of using spatial-temporal correlation compared to the correlation captured in *Video-zilla*. Therefore, we use the same intra-camera query mechanism in *Video-zilla*. For the inter-camera part, instead of dispatching queries based on the SVS similarity, we dispatch queries based on the spatial-temporal correlation between the camera that captured the image and other cameras within the system. For example, for a fire hydrant query that is captured by an in-vehicle camera in New York City, we will only search other cameras located in NYC.

Both camera-level filtering approaches achieve around 95% query *precision* performance. However, adopting spatial-temporal correlation leads to higher false negative rates (FNRs) (Figure 19), suggesting the search space is pruned too aggressively. This is because spatial-temporal correlation is a coarse-grained video similarity.

Uncertainty and error rates. Figure 19 shows the false positive rates (FPR) and false negative rates (FNR) for various queries. $Recall = 1 - FNR$. We compare indexing schemes including "classifier-only" (no indexing), per-camera top-k, spatial-temporal correlation

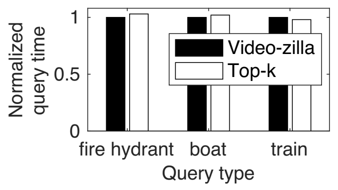


Figure 16: Bottleneck query time

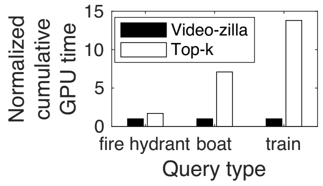
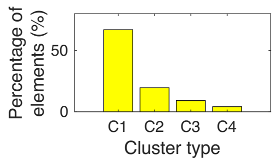
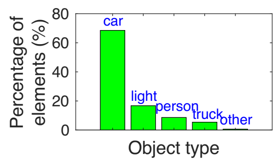


Figure 17: Total GPU time

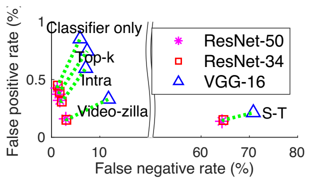


(a) Clusters in *Video-zilla*

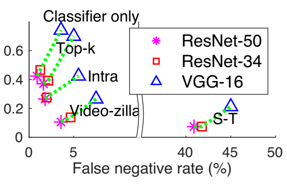


(b) Top-k index

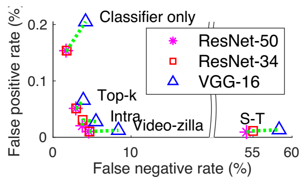
Figure 18: Feature clusters in the same video



(a) Fire hydrant



(b) Boat



(c) Train

Figure 19: Performance variation with indexing schemes and feature extractors.

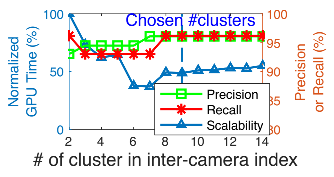


Figure 20: Tuning *Video-zilla*.

(S-T), *Video-zilla*, and a version of *Video-zilla* without the inter-camera index, where queries will be sent to all intra-camera indices. Data points corresponding to the same scheme are linked with dash lines. We experiment with three feature extractors.

When using the state-of-the-art feature extractors like ResNet-34 and ResNet-50, *Video-zilla* incurs consistent FNR losses (up to 3%) and up to 80% FPR reduction compared to *classifier only*. The FPR reduction shows *Video-zilla* prunes the search space effectively. S-T achieves similar FPRs with spatial-temporal pruning. However, that pruning is too aggressive, hence dramatically increasing the FNRs.

With the less accurate VGG-16 as the feature extractor, error rates vary by the query goal. The FNR increases for *fire hydrant* because VGG-16 classifies fire hydrants less accurately than it classifies boats and trains, which propagates to inaccurate clustering. Interestingly, *intra only* hardly suffers from increased FNR compared to the top-k index. This suggests a way to mitigate the performance disparity between query types by disabling the inter-camera index.

To summarize, the following factors affect the *Video-zilla* performance: First, the FNR is mainly constrained by the feature extractor accuracy; Second, using intra-camera indices only achieves lower FNRs, while a two-level index structure achieves lower FPRs and better processing scalability; Third, certain object types can compound the effect of inaccurate feature extractors, but reducing the number of hierarchies can mitigate the FNR disparity across object types; finally, the threshold of the decision boundary to determine query hit can be adjusted, and wider boundaries typically mean lower FNRs and higher FPRs.

Scalability and accuracy trade-off. Figure 20 shows the scalability (in terms of normalized cumulative GPU time), precision and recall trade-off for *fire hydrant* queries as the number of clusters varies within an inter-camera index for *Video-zilla*. *Boat* and *train* queries exhibit similar trade-offs. The red dashed line represents the number of clusters calculated by maximizing the silhouette value on the real-world dataset. We do not vary the cluster number in the intra-camera indices because manually setting that number causes random performance changes in the system. The *precision* generally increases and flattens around the chosen number of clusters. It is in fact convex, and will decrease when there are so many clusters that each SVS forms a distinct cluster. Conversely, *recall* is concave, and is highest when there are very few or many clusters. *Scalability* is similarly concave.

These results suggest that the precision/recall can be tuned by adjusting the number of SVSs clusters within an inter-camera index. More clusters initially increase *precision* and decrease *recall* by aggressively eliminating irrelevant video frames, including some relevant frames with few objects. The aggressive pruning also reduces the GPU time dramatically. As the cluster number increases further, pruning becomes less effective; more SVSs are examined, which increases both *recall* and the GPU time.

7.5 Case study: Specialized DNN training

A clustering query follows the format of *find all semantic video streams that are semantically similar to a video stream Y*. In this section, we highlight the potential of this novel query primitive by showcasing its application to specialized neural network training. We compare the training process based on the results of the clustering query compared to leveraging spatial-correlation among cameras. The latter is the state-of-the-art camera correlation model [42]. We use the 20 downtown in-vehicle videos as our dataset, as there is no obvious spatial-temporal correlation among others.

We adopt a method similar to the approach in [31] that only retrains the first and last three layers of a model, given limited size of the training dataset. In our case, we only select k classes of objects that cover 95% of the image frames within the whole dataset. Other classes will be labeled as an "Other" class. Each class uses 500 images as the training data.

We use four models pre-trained on ImageNet in Keras [8] as the base models: MobileNetV2, ResNet50, ResNet101, InceptionV3, which cover a range of accuracy and inference time trade-off. 50 SVSs are selected from our hierarchical index built on top of the 20 downtown videos. For *Video-zilla*, we get a list of video streams that belong to the same cluster, and use them as the training data. For spatial-correlation, we treat the videos captured within the same city as spatially-correlated. Using the output of YoloV2 as the ground truth, we compare the average top-2 classification accuracy for each trained network. *Video-zilla*'s automatic clustering is even slightly better (around 1%) than manual labeling.

By clustering SVSs based on OMD, we successfully cluster video streams that share similar classes of objects and have objects within the same class visually similar to one other. Both factors can be beneficial when training a specialized neural network. The clustering query by itself can be a novel tool to find the correlation among different video streams.

Manual labeling (tagging feeds with location) is required to leverage spatial-temporal correlation, but *Video-zilla* can automate this process by identifying semantic correlation. Further, manual labeling also implies some manual thresholding. For example, similar cars should appear in cameras at consecutive intersections or 10 exits apart on a highway. These cameras may or may not be labeled as spatially similar, but *Video-zilla* will recognize those.

7.6 Case study: Proactive video archiving

By capturing the correlation between different SVSs, *Video-zilla* can enable a proactive video archival service. For example, consider the train-station camera feed mentioned earlier. Most video frames capture empty stations which convey little information. Once a feed is divided into SVS clusters corresponding to "train arrival", "train departure" and "empty station", the "empty station" cluster can be aggressively archived. For any incoming SVS, we can estimate its potential hit rate based on the hit rate of other SVSs in the same cluster, and proactively archive low-information SVSs (for example, to some secondary storage).

For the same query objects considered earlier, we consider the ratio of total temporal length of SVSs that contain each type of queries to the overall video data length. We also take a union of all the SVSs that contain any one of the objects of interest, which emulates the case when an application needs to search for different objects. We assume each object will be queried at the same frequency, since the hit or miss behavior should not change much whether an application query for the same object once or multiple times. What matters is how many different objects are queried, not how often the same (type of) object is queried. The ratio for fire hydrant, boat, train and combined case is 1.5%, 2.0%, 26.3% and 29.1% respectively. Even considering the union query case, less than 1/3 length of video data will be retrieved. This suggests that the storage needs for the video feeds can be reduced by more than 70% by aggressively archiving low-information video segments.

8 Related work

We are not aware of previous video indexing work using the collective semantics of an entire video feed. Related work otherwise revolves around video indexing, and video analytics optimizations.

Video indexing and retrieval. There is a rich literature on content-based video indexing and retrieval [38, 49, 77], mainly on index design for specific query types, such as *shot boundary detection* [87], *key frame extraction* [50], *semantic search* [22] and spatio-temporal based retrieval [54, 66]. These leverage frame-level or feed-level video data abstractions. *Video-zilla* is orthogonal, since we focus on a new abstraction for video data, SVS, which then lends to an effective index.

Video-zilla can be seen as a form of *explanation database* [14, 69, 70, 85] for video data. Index structures have been widely used to reduce query latency in conventional SQL databases [18], key-value stores [56], graph databases [86] and many others [15]. Recent work VStore [83] and VSS [32] aim to design the storage subsystem of a video data management system (VDBMS), but still operate on the raw video data directly. As a novel abstraction to capture the inherent similarity within video data, SVS provides new ways to understand and hence efficiently index video data.

Further, existing semantic video search indices [22, 44, 46] are designed for offline video databases, where optimizations rely on the ability to process the data in multiple passes. In contrast, *Video-zilla* can work for live video feeds, where the main challenges of real-time ingestion and efficient analysis of streaming data arise from large amounts of data continuously arriving and processed in a single pass.

Video analytics systems. Thanks to advances in machine learning, recent video analytics systems have transformed video content analysis from simple information retrieval to sophisticated decision making. Some [33, 34] focus on video data management for emerging applications, like virtual reality and multi-perspective video analytics, while most [16, 17, 39, 55, 61, 75, 83, 84, 88] address the inherent video processing complexity by providing system support for video encoding and decoding and parallel and distributed processing. Understanding the semantic meaning of video data is left to upper-level applications.

Recognizing the inherent redundancy within video data, more recent efforts explore smarter video analytics approaches by understanding the video semantics. Noscope [45] and FOCUS [37] both reduce per-feed query cost, the former with cheap per-camera filters and the latter by applying a specialized and compressed DNN per video camera at ingestion. Spatula [43] optimizes cross-camera video analytics by exploiting spatial-temporal locality in a multi-camera deployment, aiming at enabling effective object tracking functionality. In contrast, the notion of semantic video streams (SVS) introduces a new dimension to characterize the correlation among video data and thus provides new optimization opportunities for object identification applications [80]. Further, all these systems implement frame analysis as part of the analytics pipeline, whereas *Video-zilla* acts as a generic indexing layer that can help simplify the development effort for new analytics applications by supporting common indexing operations and query APIs.

9 Conclusion

In this paper, we propose a notion of *semantic video stream (SVS)* that exposes the semantic content of the video feeds and captures object distribution across a set of frames. This abstraction balances the expressiveness of frame-level analysis and the efficiency of camera-level aggregation. On this basis, we design *Video-zilla*, an indexing layer interposed between a video store and analytics applications. *Video-zilla* builds a hierarchical index to capture of the correlation between SVS instances both within and across camera feeds, so as to dramatically narrow down the search space for common queries. We implement *Video-zilla* as well as proof-of-concept query case studies for object identification, video clustering, and archival. *Video-zilla* lends to almost constant scalability with the number of video feeds. We believe the notion of SVS and the hierarchical index of SVS correlation can also be applied to other domains witnessing large amounts of correlated but unstructured data.

Acknowledgments

We would like to thank all the reviewers for their helpful comments and suggestions. This work is funded by the National Science Foundation under Grant No. CNS-1815115.

References

[1] Akka toolkit. https://akka.io/.

[2] Benchmarking nearest neighbors. https://github.com/erikbern/ann-benchmarks.

[3] *Common Objects in COntext dataset.* http://cocodataset.org/home

[4] J Utah. https://www.youtube.com/channel/UCBcVQr-07MH-p9e2kRTdB3A/.

[5] Jackson Hole WebCam. https://www.seejh.com/webcams/jackson.

[6] JFastEMD. https://github.com/telmomenezes/JFastEMD.

[7] Keras, 2.3.1. https://github.com/keras-team/keras/releases/tag/2.3.1.

[8] Keras availabel models. https://keras.io/applications/.

[9] Microsoft Rocket Video Analytics Platform. https://github.com/microsoft/Microsoft-Rocket-Video-Analytics-Platform.

[10] National Oceanic and Atmospheric Administration Surveillance Market Report. https://cdn.exacq.com/auto/casestudy/pdf/a31caaeb-9a6f-f464-4590-08d97fff9828.pdf?rand=0.11452011740766466.

[11] One surveillance camera for every 11 people in Britain, says CCTV survey. https://www.telegraph.co.uk/technology/10172298/One-surveillance-camera-for-every-11-people-in-Britain-says-CCTV-survey.html.

[12] PyNNDescent. https://github.com/lmcinnes/pynndescent.

[13] Security cameras - Security, Chicago Transit Authority. https://www.transitchicago.com/security/cameras/.

[14] Firas Abuzaid, Peter Kraft, Sahaana Suri, Edward Gan, Eric Xu, Atul Shenoy, Asvin Ananthanarayan, John Sheu, Erik Meijer, Xi Wu, et al. 2018. DIFF: a relational interface for large-scale data explanation. *Proceedings of the VLDB Endowment* 12, 4 (2018), 419–432.

[15] Michael R Anderson, Michael Cafarella, German Ros, and Thomas F Wenisch. 2019. Physical representation-based predicate optimization for a visual analytics database. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1466–1477.

[16] Lixiang Ao, Liz Izhikevich, Geoffrey M Voelker, and George Porter. 2018. Sprocket: A serverless video processing framework. In *Proceedings of the ACM Symposium on Cloud Computing*. 263–274.

[17] Favyen Bastani, Songtao He, Arjun Balasingam, Karthik Gopalakrishnan, Mohammad Alizadeh, Hari Balakrishnan, Michael Cafarella, Tim Kraska, and Sam Madden. 2020. MIRIS: fast object track queries in video. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1907–1921.

[18] Rudolf Bayer and Edward McCreight. 2002. Organization and maintenance of large ordered indexes. In *Software pioneers*. Springer, 245–262.

[19] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. 2010. Moa: Massive online analysis. *Journal of Machine Learning Research* 11, May (2010), 1601–1604.

[20] blessedhomosapiens. American Roads. https://www.youtube.com/user/blessedhomosapiens/videos.

[21] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[22] Shih-Fu Chang, Wei-Ying Ma, and Arnold Smeulders. 2007. Recent advances and challenges of semantic image/video search. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, Vol. 4. IEEE, IV–1205.

[23] Edgar Chávez and Gonzalo Navarro. 2003. Probabilistic proximity search: Fighting the curse of dimensionality in metric spaces. *Inform. Process. Lett.* 85, 1 (2003), 39–46.

[24] Sahil Chinoy. 2019. We Built an 'Unbelievable' (but Legal) Facial Recognition Machine. https://www.nytimes.com/interactive/2019/04/16/opinion/facial-recognition-new-york-city.html

[25] Paolo Ciaccia, Marco Patella, and Pavel Zezula. 1997. M-tree: An efficient access method for similarity search in metric spaces. In *Vldb*, Vol. 97. Citeseer, 426–435.

[26] Albert Clapés, Miguel Reyes, and Sergio Escalera. 2013. Multi-modal user identification and object recognition surveillance system. *Pattern Recognition Letters* 34, 7 (2013), 799–808.

[27] Robert T Collins, Alan J Lipton, Hironobu Fujiyoshi, and Takeo Kanade. 2001. Algorithms for cooperative multisensor surveillance. *Proc. IEEE* 89, 10 (2001), 1456–1477.

[28] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to algorithms.* MIT press.

[29] Georgios Diamantopoulos and Michael Spann. 2005. Event detection for intelligent car park video surveillance. *Real-Time Imaging* 11, 3 (2005), 233–243.

[30] Wei Dong, Charikar Moses, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*. 577–586.

[31] 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*. 123–136.

[32] Brandon Haynes, Maureen Daum, Dong He, Amrita Mazumdar, Magdalena Balazinska, Alvin Cheung, and Luis Ceze. 2021. Vss: A storage system for video analytics. In *Proceedings of the 2021 International Conference on Management of Data*. 685–696.

[33] Brandon Haynes, Maureen Daum, Amrita Mazumdar, Magdalena Balazinska, Alvin Cheung, and Luis Ceze. 2020. VisualWorldDB: A DBMS for the Visual World.. In *CIDR*.

[34] Brandon Haynes, Amrita Mazumdar, Magdalena Balazinska, Luis Ceze, and Alvin Cheung. 2018. Lightdb: A dbms for virtual reality video. *Proceedings of the VLDB Endowment* 11, 10 (2018).

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[36] Katherine A Heller and Zoubin Ghahramani. 2005. Bayesian hierarchical clustering. In *Proceedings of the 22nd international conference on Machine learning*. 297–304.

[37] 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)*. 269–286.

[38] Weiming Hu, Nianhua Xie, Li Li, Xianglin Zeng, and Stephen Maybank. 2011. A survey on visual content-based video indexing and retrieval. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 41, 6 (2011), 797–819.

[39] Qi Huang, Petchean Ang, Peter Knowles, Tomasz Nykiel, Iaroslav Tverdokhlib, Amit Yajurvedi, Paul Dapolito IV, Xifan Yan, Maxim Bykov, Chuen Liang, et al. 2017. SVE: Distributed video processing at Facebook scale. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 87–103.

[40] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. 2018. Videoedge: Processing camera streams using hierarchical clusters. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 115–131.

[41] IHS. Top Video Surveillance-Trends-2018. https://cdn.ihs.com/www/pdf/Top-Video-Surveillance-Trends-2018.pdf.

[42] Samvit Jain, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, and Joseph Gonzalez. 2019. Scaling video analytics systems to large camera deployments. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*. 9–14.

[43] Samvit Jain, Xun Zhang, Yuhao Zhou, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Paramvir Bahl, and Joseph Gonzalez. 2020. Spatula: Efficient cross-camera video analytics on large camera networks. In *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 110–124.

[44] Daniel Kang, Peter Bailis, and Matei Zaharia. 2018. BlazeIt: optimizing declarative aggregation and limit queries for neural network-based video analytics. *arXiv preprint arXiv:1805.01046* (2018).

[45] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. Noscope: optimizing neural network queries over video at scale. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1586–1597.

[46] Daniel Kang, John Guibas, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2020. Task-agnostic Indexes for Deep Learning-based Queries over Unstructured Data. *arXiv preprint arXiv:2009.04540* (2020).

[47] Ari Kobren, Nicholas Monath, Akshay Krishnamurthy, and Andrew McCallum. 2017. A hierarchical algorithm for extreme clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 255–264.

[48] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[49] Michael S Lew, Nicu Sebe, Chabane Djeraba, and Ramesh Jain. 2006. Content-based multimedia information retrieval: State of the art and challenges. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 2, 1 (2006), 1–19.

[50] Xuelong Li, Bin Zhao, and Xiaoqiang Lu. 2017. Key frame extraction in the summary space. *IEEE transactions on cybernetics* 48, 6 (2017), 1923–1934.

[51] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. 2017. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2117–2125.

[52] Haibin Ling and Kazunori Okada. 2007. An efficient earth mover's distance algorithm for robust histogram comparison. *IEEE transactions on pattern analysis and machine intelligence* 29, 5 (2007), 840–853.

[53] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge assisted real-time object detection for mobile augmented reality. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–16.

[54] Xiaochen Liu, Pradipta Ghosh, Oytun Ulutan, BS Manjunath, Kevin Chan, and Ramesh Govindan. 2019. Caesar: cross-camera complex activity recognition. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. 232–244.

[55] Yao Lu, Aakanksha Chowdhery, and Srikanth Kandula. 2016. Optasia: A relational platform for efficient large-scale video analytics. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*. 57–70.

[56] Yandong Mao, Eddie Kohler, and Robert Tappan Morris. 2012. Cache craftiness for fast multicore key-value storage. In *Proceedings of the 7th ACM european*

conference on Computer Systems. ACM, 183–196.

[57] Daniel Müllner. 2011. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378* (2011).

[58] Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, JK Aggarwal, Hyungtae Lee, Larry Davis, et al. 2011. A large-scale benchmark dataset for event recognition in surveillance video. In *CVPR 2011*. IEEE, 3153–3160.

[59] Ofir Pele and Michael Werman. 2008. A linear time histogram metric for improved sift matching. In *European conference on computer vision*. Springer, 495–508.

[60] Ofir Pele and Michael Werman. 2009. Fast and robust earth mover's distances. In *2009 IEEE 12th International Conference on Computer Vision*. IEEE, 460–467.

[61] Alex Poms, Will Crichton, Pat Hanrahan, and Kayvon Fatahalian. 2018. Scanner: Efficient video analysis at scale. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 138.

[62] Virtual Railfan. Kansas City East. https://www.youtube.com/watch?v=WUBQJosNm8o.

[63] Virtual Railfan. San Juan Capistrano, California USA - Virtual Railfan LIVE. https://www.youtube.com/watch?v=zXqx56hlpd4.

[64] Tomi D Räty. 2010. Survey on contemporary remote surveillance systems for public safety. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40, 5 (2010), 493–515.

[65] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7263–7271.

[66] Wei Ren, Sameer Singh, Maneesh Singh, and YS Zhu. 2009. State-of-the-art on spatio-temporal information-based video retrieval. *Pattern Recognition* 42, 2 (2009), 267–282.

[67] Bernhard Rinner. 2019. Can We Trust Smart Cameras? *Computer* 52, 5 (2019), 67–70.

[68] Peter J Rousseeuw. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics* 20 (1987), 53–65.

[69] Sudip Roy, Arnd Christian König, Igor Dvorkin, and Manish Kumar. 2015. Perfaugur: Robust diagnostics for performance anomalies in cloud services. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 1167–1178.

[70] Sudeepa Roy and Dan Suciu. 2014. A formal approach to finding explanations for database queries. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 1579–1590.

[71] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. 1998. A metric for distributions with applications to image databases. In *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*. IEEE, 59–66.

[72] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. 2000. The earth mover's distance as a metric for image retrieval. *International journal of computer vision* 40, 2 (2000), 99–121.

[73] V Ruzicka and Franz Franchetti. 2018. Fast and accurate object detection in high resolution 4K and 8K video using GPUs. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 1–7.

[74] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 815–823.

[75] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.

[76] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[77] Cees GM Snoek, Marcel Worring, et al. 2009. Concept-based video retrieval. *Foundations and Trends® in Information Retrieval* 2, 4 (2009), 215–322.

[78] Alpha Tech. Werkhaven Urk. https://www.youtube.com/watch?v=IebzjqLQoYg.

[79] Michel Verleysen and Damien François. 2005. The curse of dimensionality in data mining and time series prediction. In *International work-conference on artificial neural networks*. Springer, 758–770.

[80] Harald Vogt. 2002. Efficient object identification with passive RFID tags. In *International Conference on Pervasive Computing*. Springer, 98–113.

[81] WebCamNL. WebCam.NL | www.hollandmarinas.com | live FULL HD PTZ camera. https://www.youtube.com/watch?v=7g0lWkV61lY&feature=emb_logo.

[82] Mengwei Xu, Tiantu Xu, Yunxin Liu, Xuanzhe Liu, Gang Huang, and Felix Xiaozhu Lin. 2019. Supporting Video Queries on Zero-Streaming Cameras. *arXiv preprint arXiv:1904.12342* (2019).

[83] Tiantu Xu, Luis Materon Botelho, and Felix Xiaozhu Lin. 2019. Vstore: A data store for analytics on large videos. In *Proceedings of the Fourteenth EuroSys Conference 2019*. 1–17.

[84] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. 2017. Lavea: Latency-aware video analytics on edge computing platform. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 15.

[85] Dong Young Yoon, Ning Niu, and Barzan Mozafari. 2016. Dbsherlock: A performance diagnostic tool for transactional databases. In *Proceedings of the 2016 International Conference on Management of Data*. 1599–1614.

[86] Dayu Yuan and Prasenjit Mitra. 2013. Lindex: a lattice-based index for graph databases. *The VLDB Journal—The International Journal on Very Large Data Bases* 22, 2 (2013), 229–252.

[87] Jinhui Yuan, Huiyi Wang, Lan Xiao, Wujie Zheng, Jianmin Li, Fuzong Lin, and Bo Zhang. 2007. A formal study of shot boundary detection. *IEEE transactions on circuits and systems for video technology* 17, 2 (2007), 168–186.

[88] Tan Zhang, Aakanksha Chowdhery, Paramvir 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*. 426–438.

[89] Wuyang Zhang, Zhezhi He, Luyang Liu, Zhenhua Jia, Yunxin Liu, Marco Gruteser, Dipankar Raychaudhuri, and Yanyong Zhang. 2021. Elf: accelerate high-resolution mobile deep vision with content-aware parallel offloading. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 201–214.

[90] Wei Zhang, QM Jonathan Wu, and Hai bing Yin. 2010. Moving vehicles detection based on adaptive motion histogram. *Digital Signal Processing* 20, 3 (2010), 793–805.

[91] Zhongna Zhou, Xi Chen, Yu-Chia Chung, Zhihai He, Tony X Han, and James M Keller. 2008. Activity analysis, summarization, and visualization for indoor human activity monitoring. *IEEE Transactions on Circuits and Systems for Video Technology* 18, 11 (2008), 1489–1498.