Visual Road: A Video Data Management Benchmark

Brandon Haynes, Amrita Mazumdar
Magdalena Balazinska, Luis Ceze, Alvin Cheung
{bhaynes,amrita,magda,luisceze,akcheung}@cs.washington.edu
Paul G. Allen Center for Computer Science
University of Washington

http://visualroad.uwdb.io

ABSTRACT

Recently, video database management systems (VDBMSs) have re-emerged as an active area of research and development. To accelerate innovation in this area, we present Visual Road, a benchmark that evaluates the performance of these systems. Visual Road comes with a data generator and a suite of queries over cameras positioned within a simulated metropolitan environment. Visual Road's video data is *automatically* generated with a high degree of realism, and annotated using a modern simulation and visualization engine. This allows for VDBMS performance evaluation while scaling up the size of the input data. Visual Road is designed to evaluate a broad variety of VDBMSs: real-time systems, systems for longitudinal analytical queries, systems processing traditional videos, and systems designed for 360° videos. We use the benchmark to evaluate three recent VDBMSs both in capabilities and performance.

KEYWORDS

Benchmarking and performance evaluation; multimedia databases; video data management; virtual reality video

ACM Reference Format:
Brandon Haynes, Amrita Mazumdar and Magdalena Balazinska, Luis Ceze, Alvin Cheung. 2019. Visual Road: A Video Data Management Benchmark. In *2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3299869.3324955

SIGMOD '19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5643-5/19/06…$15.00
https://doi.org/10.1145/3299869.3324955

(a) Rain with dense clouds

(b) Overcast skies at sunset

Figure 1: Two frames from Visual Road cameras illustrate the realism and variety of the benchmark. Some depicted assets copyright [13] and [14]. Complete videos available at visualroad.uwdb.io/datasets.

1 INTRODUCTION

Video data management has recently re-emerged as an active research area due to advances in machine learning and graphics hardware, as well as the emergence of applications such as adaptive streaming and virtual reality. This technology push and application pull have led the community to develop many new systems to efficiently process and manage video data [1, 20, 26, 32, 33, 35, 44].

Existing systems quantify their performance by reporting their efficiency when processing various ad hoc workloads both in terms of the input videos selected and the executed queries. However, comprehensive and easily reproducible system comparisons are missing. A key challenge is that there is currently no clear way to reliably and objectively benchmark performance among the various

recently proposed video database management systems (VDBMSs). This deficiency is due to a lack of: (i) a robust, sufficiently-complex video dataset (in terms of resolution, quantity, duration, and variety of content); and (ii) an architecture-agnostic specification of a common set of queries that may be executed on current and future VDBMSs.

Analogous to standardized benchmarks for other areas of data management research (such as transaction [53] and analytical processing [55]), any benchmark for video data processing needs the ability to test systems at different scales and in a repeatable manner. To achieve this, short video segments or ad hoc video streams will not suffice. However, as shown in Table 1, such inputs have been typical in the evaluation of recent VDBMSs (with the exception of Scanner [44], which comes with multiple video streams but has no way to scale input data size). While such data may be used in isolation to evaluate properties such as prediction accuracy or query planning performance of a specific system, it does not allow conclusions about VDBMS performance (e.g., in terms of throughput) when executing thousands of such queries. It also complicates the comparison of the various systems in terms of the types of queries they support.

To address this deficiency, we develop a new benchmark aimed specifically at VDBMSs. As we describe in Section 2, evaluating video database management systems poses a number of unique challenges that do not arise in data management benchmarks for other domains. For example, using randomized inputs—as is common in benchmarks for relational database systems—is infeasible due to the inherent semantic structure of video, as algorithms such as motion detection require consecutive video frames to contain coherent features.

Equally difficult is determining whether a VDBMS produces a correct answer to a query, which requires accurate ground truth. Existing datasets are manually annotated with this metadata, but manually annotating the hundreds of thousands of inputs required to evaluate performance under load, scalability, and other similar metrics is impractical.

To address these challenges, we develop Visual Road[1], a benchmark designed to evaluate the performance of VDBMSs in the face of a diverse query workload. Visual Road reproducibly and objectively measures how well a VDBMS executes a battery of video-oriented workloads. Visual Road includes a set of evaluation queries and a data generator. The queries are divided into "microbenchmark" operations that test isolated features found in current VDBMSs, along with larger "composite" queries that measure a VDBMS's ability to execute typical end-to-end applications drawn from the recent literature.

[1]Name inspired by Linear Road [2], a benchmark for streaming database management systems.

Table 1: Many recent video database systems evaluate using only a small number of distinct inputs. In Section 6 we evaluate the subset that have source available and can be installed on our hardware.

Name	# Distinct Inputs
Optasia [35]	3
LightDB [20]	4
Chameleon [31]	5
BlazeIt [32]	6
NoScope [33]	7
Focus [25]	14
Scanner [44]	>100

Visual Road comes with a data generator that produces input videos for the benchmark. To allow the creation of a virtually unlimited number of these input videos, Visual Road uses a modern simulation, visualization, and gaming engine [13] to deterministically generate realistic videos within a simulated metropolitan world (see Figure 1). Visual Road allows users to vary the city size, number of cameras, and length of video in its simulation to arbitrarily large sizes. Additionally, its simulation allows for the *automatic* calculation of precise ground truth and other metadata about generated videos, without the need for manual annotation. Finally, the cameras used in Visual Road are extremely flexible. In addition to generating ordinary two-dimensional video, they can also produce more complex video types (e.g., panoramic 360° video) that are used with a more complex category of virtual reality (VR)-oriented benchmark queries. Overall, Visual Road's generated video datasets are rich and highly realistic. They can serve to execute various real-world applications such as vehicle tracking and compute meaningful results. The queries provided with the benchmark include a variety of both simple queries and complex applications to exercise benchmarked systems along various dimensions.

Visual Road is designed to be implementable across a wide variety of VDBMS architectures, including those that perform video querying at scale (e.g., Scanner [44], Optasia [35], Chameleon [31]), operate on emerging forms of video data (e.g., LightDB [20]), and perform deep learning inference (e.g., NoScope [33], BlazeIt [32], Focus [25]). In the same way that relational database systems target subsets of benchmarks (e.g., a specific TPC query), Visual Road is designed to be flexible: a user may either select specific applicable queries or groups of queries appropriate for their systems or execute the entire benchmark to demonstrate broad functionality.

Visual Road is also extensible, such that future innovations and workload types can be easily incorporated into subsequent versions. This includes both the ability to introduce new and unexpected elements into the video simulation, and also to increase the complexity of the benchmark queries (e.g., by increasing the number of cameras, range of benchmark parameters, or available machine learning algorithms).

Each benchmark query is specified in a VDBMS-agnostic manner adaptable to a wide variety of VDBMS types and architectures. To illustrate this wide applicability, we have implemented the benchmark on three recent VDBMSs. We discuss how Visual Road leads to objective and reproducible results that can serve as a fair comparison between VDBMSs.

In summary, we make the following contributions:

- We develop Visual Road, a benchmark for video database management systems. The benchmark includes a data generator, along with microbenchmark and composite queries (Sections 3 to 5).
- We show that the videos generated by Visual Road produce results of sufficient quality to match existing, manually-curated video datasets (Section 6).
- We implement Visual Road on several VDBMSs and show how it helps to compare the systems both in terms of capabilities and performance (Section 6).

The remainder of this paper is organized as follows. We first describe the challenges associated with evaluating VDBMSs at scale (Section 2). We then introduce the Visual Road benchmark (Sections 3 and 4) and describe its implementation (Section 5). We evaluate the quality of the benchmark videos, the data generation performance, and the ability of the benchmark to compare three modern VDBMSs (Section 6). We conclude with related work and future directions.

2 CHALLENGES

Designing a benchmark that targets VDBMSs poses several unique challenges not found in existing database benchmarks. These include:

Creating sufficiently large datasets. Many existing video corpora (e.g., VIRAT [40], UA-DETRAC [56], PETS04 [17], Okutama-Action [5]) consist of a modest amount ($<$ 30 hours) of curated videos. Such videos cannot sufficiently scale without massive duplication and redundancy. As we show in Section 6, executing video workloads with a small number of distinct inputs (reasonably) allows systems to optimize (e.g., by aggressive caching) in ways that would not be possible with a large number of video inputs. This optimization leads to performance results that are skewed and impair comparisons between systems. While duplicating or randomly generating video might avoid this issue, these approaches suffer from other challenges that we describe below (and show to be problematic in Section 6).

Manual annotation. Existing video datasets are typically hand-annotated with ground truth and other metadata. Prior work has utilized such annotations to measure system performance (e.g., classification accuracy). This manual "human-in-the-loop" process does not scale well to allow for the large datasets that are needed to evaluate system performance at scale.

Semantic structure. Randomly-generated datasets, which are commonly used in relational DBMS benchmarks (e.g., TPC-C [53], TPC-H [55], TPC-E [54]), are ill-suited for video-oriented workloads. Videos are inherently highly-structured (e.g., pixels in adjacent video frames are highly correlated), and both video encoders and VDBMSs take advantage of these characteristics to speed up processing. For instance, encoders exploit redundancies in adjacent video frames for efficient compression. Because of this, a benchmark that relies on noise or other randomly-generated inputs will produce unrealistic results.

No universal language or functionality. To date, no standard query language, data model, or functionality exists across all VDBMSs (however, some work has recently explored this space [20]). Expressing a set of queries that can be implemented and executed across a wide variety of VDBMSs remains a challenge.

Visual Road is designed to address the challenges described above. Its simulation-oriented approach lets users to place an arbitrary number of cameras, each with configurable position, resolution, and field of view. This configurability allows for the generation of a practically unlimited number of input videos. Further, the resulting corpus is realistic—for example, videos contain semantically-valid objects (e.g., cars, buildings, pedestrians), and cameras with overlapping fields of view both show the same objects (albeit from differing perspectives).

Visual Road's approach also allows for automatic generation of ground truth and other metadata. If a VDBMS query result indicates that a pedestrian is present in frame i of video j, Visual Road is able to evaluate the geometry of the scene that produced the video and automatically determine whether this result is correct.

Finally, we have taken care to express each query in a VDBMS- and architecture-agnostic manner. VDBMSs are free to implement each such query in any manner is appropriate for that system.

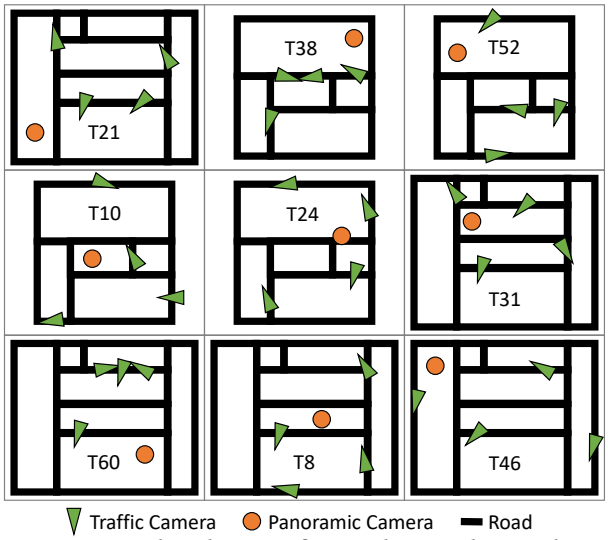

Figure 2: Overhead view of a randomized Visual Road configuration with $L = 9$.

3 THE VISUAL ROAD BENCHMARK

Video used for the Visual Road benchmark is generated in *Visual City*, a pseudorandomly-generated, simulated metropolitan area. Visual City currently contains road networks, vehicles, pedestrians, landscaping, buildings, bridges, traffic, ground-based cameras, and other features found in real-world cities. Visual City is also affected by a number of conditions such as cloud cover, precipitation, and sun position. Sample photos taken from Visual City are shown in Figure 1. We posit that this environmental complexity is both important and sufficient to ensure that benchmarked VDBMSs are exercised in interesting ways. The features of the generated city could also be extended in subsequent versions of the benchmark (e.g., by incorporating wildlife, tunnels, or lakes) to increase the complexity, variety, and unexpectedness of the simulation.

3.1 Benchmark Data

As shown in Figure 2, Visual City is laid out as a disconnected set of tiles. Each tile is drawn uniformly with replacement from a pool of tiles associated with a particular version of Visual Road. The version described in this paper contains 72 tiles (see Section 5) and each tile is several square kilometers in size. Each tile contains different weather conditions, pedestrian and vehicle densities, and geography.

Video data is captured in Visual City via a number of cameras. Each tile is associated with a camera configuration C that specifies various types and numbers of cameras. To monitor traffic conditions, each tile contains c_t randomly-oriented *traffic cameras* positioned 10-20 meters above a roadway, along with c_p randomly-oriented *panoramic cameras* positioned 5-10 meters above sidewalks. Each panoramic camera is composed of four ordinary cameras with overlapping 120°fields of view positioned so that they overlap to capture a 360°field of view. The current Visual Road prototype sets $C = \{c_t, c_p\} = \{4, 1\}$ for each tile.

When generating video data, a user provides values for four hyperparameters. A *scale factor* L determines the number of tiles in the city. A user also selects a *resolution* (e.g., 3840×2160) and a simulation *duration* that is globally applied to each of the cameras. Finally, a random *seed* s allows other users to deterministically reproduce datasets by reinitializing the pseudorandom number generator with the same seed.

In addition to providing city size, the scale factor also determines the number of queries generated throughout a Visual City. The Visual Road prototype currently generates $4L$ queries for each type detailed in this section. This batch size allows for a reasonable balance between dataset generation time and benchmark execution time.

The *Visual City Generator (VCG)* is used to generate input videos captured within Visual City. It accepts the hyperparameters described above and uses these values to construct a Visual City. First, it randomly chooses L tiles from the available set of tiles (with replacement). Each tile is configured and populated using a tile-specific configuration (e.g., pedestrians and vehicles are randomly spawned in number and locations specific to that tile). Cameras are then randomly positioned in each tile subject to the constraints described above. The VCG then executes the simulation and captures videos generated by each camera. These videos are encoded using the H264 codec [51] and stored as flat files (see Section 6). The VCG also generates additional supporting metadata required for verifying the results of specific queries (e.g., Q6 in Section 4).

A VDBMS reports performance by executing the benchmark using videos captured in a Visual City as input. The benchmark comes with a set of pregenerated datasets for immediate use (see Table 2); users may report results using these datasets when comparing to other systems (e.g., "We evaluate using version 1.0 of the 4K-SHORT dataset"). Alternatively, users may deterministically generate their own datasets (see below) and report the configuration along the results (e.g., "We generated and executed the Visual Road 1.0 benchmark using scale L, resolution R, duration t, and seed s"). By using the same configuration, competing VDBMSs can reproduce the identical dataset and compare results.

In this first version of the benchmark, we choose to only allow users to adjust the scale factor, resolution, duration, and seed parameters. This choice keeps the benchmark easy to use. If the community finds it useful, however, future versions could easily expose other parameters, such as testing only on tiles with sunny weather or changing the density of the cameras in individual tiles.

Table 2: Pregenerated datasets available at visualroad.uwdb.io/datasets.

Name	Hyperparameters
1K-SHORT	$\{L = 2, R = 960 \times 540, t = 15 \text{ min}\}$
1K-LONG	$\{L = 4, R = 960 \times 540, t = 60 \text{ min}\}$
2K-SHORT	$\{L = 2, R = 1920 \times 1080, t = 15 \text{ min}\}$
2K-LONG	$\{L = 4, R = 1920 \times 1080, t = 60 \text{ min}\}$
4K-SHORT	$\{L = 2, R = 3840 \times 2160, t = 15 \text{ min}\}$
4K-LONG	$\{L = 4, R = 3840 \times 2160, t = 60 \text{ min}\}$

3.2 Benchmark Execution

A VDBMS can execute the benchmark either *offline* or *online*. Offline processing simulates batch processing of historical video streams, where the VDBMS has random access to entire video files on persistent storage. Online processing simulates real-time video processing, where data is exposed via a forward-only iterator with unknown total duration.

A separate *Visual City Driver (VCD)* is provided with the benchmark and is responsible for reading the input videos, exposing encoded video data to a VDBMS, submitting queries to the VDBMS being measured, and evaluating the correctness of a VDBMS's query results.

When benchmarking in online mode, a VDBMS may access each video using either a named pipe (on a single local file system) or via the RTP protocol [49]. In this mode, video data is throttled to a simulated real-time throughput (i.e., the VCD exposes video frames at the corresponding camera's capture rate). The VCD blocks on attempts to read video data beyond this rate. For a VDBMS benchmarking in offline mode, the VCD additionally ensures each input video is available on the local file system (on a single node, if the VDBMS supports distributed execution) or a distributed file system (we currently support HDFS).

The VCD uses the scale factor to simulate submission of simultaneous instances (a "query batch") of each benchmark query to the VDBMS. The VCD submits batches in benchmark query order (i.e., Q1 is submitted before Q2). A VDBMS may execute each batch in a manner that is most performant (e.g., serially or in parallel), and may optionally quiesce or restart upon completing a batch.

Each benchmark query is a template with one or more parameters (see Table 3). The VCD creates each instance in a query batch by assigning values selected uniformly at random for each parameter from their respective domains. The VDBMS is only responsible for executing the query instance, and does not participate in selecting the parameter values.

A VDBMS may do one of two things with the H264- or HEVC-encoded result of a query. First, in *write mode*, as a VDBMS completes each instance in a query batch, it should write the result to a VCD-specified location on the local file system (or on a VCD-specified node for distributed systems) so the VCD can verify the correctness of each output. In this mode the time to persist results is included in the total execution time for the query batch. Alternatively, *streaming mode* allows a VDBMS to discard the results of a query and avoid the write overhead. However, in this mode a user must ensure that the results of the queries are correct, either by executing a second time in write mode or by doing so manually. We show in Section 6.4 that the performance differences between these two modes are negligible.

Finally, the VCD also validates the correctness of the results generated by a VDBMS. Depending on the query, it does so either by *frame validation*, which compares VDBMS output videos to reference output videos, or *semantic validation*, which compares a query result with the actual scene geometry used in its input(s). In Visual Road, most microbenchmark queries are verified using frame validation. For these queries, the VCD executes its reference implementation and compares each frame with the VDBMS's output using a *validation metric*. While future versions of Visual Road may allow for different metrics, the one used in the present version is the peak signal-to-noise ratio (PSNR). The PSNR is a frequently-used image comparison metric, and values $\geq$ 40 dB are considered to be near-lossless [23, 28]. Visual Road adopts this threshold as a cutoff for validation.

Query Q2(c) and Q2(d) are verified using semantic validation. In this case, the VCD compares a VDBMS's response to the actual objects that were present in the frames used as input to the query. For example, if a VDBMS indicates a car i is present in frame j, the VCD queries the simulation engine to determine if car i was visible to the camera at the instant the frame was captured.

When reporting results, an evaluator must report validation descriptive statistics for each query. For queries executed in online mode, this should be reported in frames per second. A VDBMS executing offline analytical queries should report total query runtime or frames per second. The evaluator should also report other global elections such as scale factor, resolution, duration, and execution mode.

4 VISUAL ROAD QUERIES

The Visual Road benchmark aims to evaluate VDBMS performance by executing a varied workload. It does so by measuring performance using microbenchmark (Table 5) and composite macrobenchmark queries (Section 4.2). Microbenchmark queries target the performance of individual VDBMS operations in isolation. Each microbenchmark involves a single, basic operation exposed by recent VDBMSs that are common in video applications. Composite queries, drawn from recent literature (see Section 7), utilize two or more microbenchmarks to implement more complex tasks.

Table 3: Microbenchmark parameters and domains.

Query	Parameter	Domain
Q1	x_1, x_2	$0 \le x_1 < x_2 \le R_x$
	y_1, y_2	$0 \le y_1 < y_2 \le R_y$
	t_1, t_2	$0 \le t_1 < t_2 \le D$
Q2(b)	d	[3..20]
Q2(c)	A	YOLO [46]
	O	{Pedestrian, Vehicle}
Q2(d)	m	[2..60]
	ϵ	(0, 1)
Q3	Δx	$\{R_x/2^n \mid n \in [1..3]\}$
	Δy	$\{R_y/2^n \mid n \in [1..3]\}$
	b_i	$\{2^n, n \in [16..22]\}$
Q4, Q5	α	$\{2^n \mid n \in [1..5]\}$
	β	$\{2^n \mid n \in [1..5]\}$

A VDBMS individually measures its performance for each query. As detailed previously, for a given query Q_i, the VCD uses the scale factor L to submit a query batch containing $4L$ instances of Q_i to the VDBMS. The free parameters for each instance Q_i^j, summarized in Table 3, are uniformly selected (by the VCD) at random from their domain. Below we describe each microbenchmark query. Each query operates on a randomly-selected input video.

Several queries include ML-based computer vision algorithms, such as object detection. The benchmark requires that all VDBMSs use specified, state-of-the-art algorithms, and focuses on evaluating the execution performance of queries that need to apply those algorithms rather than their quality. For the same reason, the benchmark videos do not purposefully include unusual scenarios designed to challenge computer vision methods. In case query accuracy or algorithm selection becomes a concern, users of the benchmark could be required to publish the F1 scores of their query results.

4.1 Microbenchmarks

The following microbenchmark queries, formally defined in Table 5, measure a VDBMS's ability to repeatedly perform small operations over input videos. Each is defined in terms of familiar database operations (e.g., load, window, aggregate) along with the convenience operations defined in Table 4. Each video is composed of a sequence of *frames* that are temporal samples of visual data with resolution $R_x \times R_y$. At coordinate (x, y) of each frame is a *pixel* that contains a color in a color space (e.g., RGB, YUV). Each executing VDBMS stores the query result on disk (when executed in write mode), or streams the result (when executed in streaming mode). In Section 4.2 we compose many of these microbenchmarks to form more substantial, real-world applications drawn from recent literature.

Table 4: Convenience operators used in microbenchmark queries.

Name	Type
PMap	$video \to (pixel \to pixel) \to video$ Map over individual pixels in a video
FMap	$video \to (frame \to frame) \to video$ Map over video frames
JoinP	$video \to video \to (pixel \to pixel \to pixel) \to video$ Join (by pixel coordinate) over video inputs and apply a projection on each joined pair
Interpolate	$video \to (frame \to \mathbb{N}^2 \to frame) \to \mathbb{N}^2 \to video$ Interpolate a video to a new resolution
Sample	$video \to \mathbb{N}^2 \to video$ Downsample a video at a lower resolution

4.1.1 *Video Manipulation.*

Spatial & Temporal Selection (Q1). A VDBMS must be able to efficiently spatially and temporally select subregions of videos. This ability is exercised, for example, in applications that select highlights containing relevant data, construct cinematographic montages, or apply object detection to a region of interest. Query Q1 measures a VDBMS's ability to perform this type of operation.

Given a cropping rectangle bounded by the respective upper-left and lower-right points (x_1, y_1) and (x_2, y_2) and a temporal range (t_1, t_2), query Q1 crops the frames and duration of a random input video V_i. The cropping rectangle points and temporal range are chosen uniformly at random.

Transformation (Q2) & Subquery (Q3). A VDBMS must be able to efficiently perform transformations at various granularities (e.g., per-pixel, using a stencil, over regions, and for entire frames). Queries Q2 and Q3 test a VDBMS's ability to transform input videos at these scales.

The first transformation, Q2(a), requires that a VDBMS convert a video to grayscale. The VCD reference implementation does this by dropping chroma information (i.e., the U and V channels in YUV color space) and leaves luminescence (Y) unchanged.

Query Q2(b) performs a Gaussian blur convolution [48] over an input video by applying a $d \times d$ kernel over the pixels of each video. It does so by invoking a user-defined function *blur* that is parameterized by the kernel size.

Next, query Q2(c) generates rectangular bounding boxes for objects in an input video. It does so by applying an object-detection algorithm A to each input video frame (in the present version A is a singleton consisting of the YOLO [46] algorithm). This algorithm associates each pixel p_i in each frame with zero or more object classes $O = \{o_1, ..., o_n\}$. The VDBMS associates a constant color c_j with each class o_j and a "null" black color ω for regions not associated with any class.

Table 5: Visual Road microbenchmark queries. See Table 4 for the types of non-standard database operations. The function ω-coalesce is defined in Equation 1.

#	Name	Pseudocode
Q1	**Select**	`Load`(V_i)`.Select`($x_1 \le x \le x_2$, $y_1 \le y \le y_2$, $t_1 \le t \le t_2$) Crop video frames to a rectangle at (x_1, y_1) and (x_2, y_2) and between time t_1 and t_2.
Q2	**Transform**	
(a)	Grayscale	`Load`(V_i)`.PMap`(f) Convert a video to grayscale using f that takes in a YUV pixel (y, u, v) and returns $(y, 0, 0)$.
(b)	Blur	f = *blur*(d); `Load`(V_i)`.FMap`(f) *blur* generates a $d \times d$ Gaussian convolution function f, which is applied to each frame of a video.
(c)	Boxes	f = *boxes*(A, O); `Load`(V_i)`.FMap`(f) *boxes* returns a function f that identifies object classes O using algorithm A. f is applied to the video to produce boxes for detected instances.
(d)	Masking	w is a func. that creates a window of m points. a is a func. that computes the mean of m pixels. p is a func. that takes in two pixels p_v, p_b and returns ω if $\lvert\frac{p_v - p_b}{p_v}\rvert < \epsilon$, or p_v otherwise. B = `Load`(V_i)`.Window`(w)`.Aggregate`(a) `Load`(V_i)`.JoinP`(B, p) Apply an m-frame mean-filter to each frame in a video, and set pixels below threshold ϵ to ω.
Q3	**Subquery**	f = *encoder*(B) `Load`(V_i)`.Partition`($\Delta x, \Delta y$)`.Subquery`(f) Cut each video frame into tiles of size $(\Delta x, \Delta y)$ and re-encode each tile using f at bitrate B, as generated by *encoder*.
Q4	**Upsample**	`Load`(V_i)`.Interpolate`(*bilin*, $\alpha R_x, \beta R_y$) Upsample each frame to size $(\alpha R_x, \beta R_y)$ using bilinear interpolation.
Q5	**Downsample**	`Load`(V_i)`.Sample`($^{R_x}/_{\alpha}, ^{R_y}/_{\beta}$) Reduce each video frame to size $(\frac{R_x}{\alpha}, \frac{R_y}{\beta})$.
Q6	**Union**	
(a)	Boxes	B = $Q_{2c}(V_i)$ `Load`(V_i)`.JoinP`(B, ω-coalesce) Overlay bounding rectangles B (see Q2c) on top of an input video V_i.
(b)	Captions	`Load`(V_i)`.JoinP(Load`(C_i)`,`ω-coalesce`)` Overlay captions defined in C_i on top of an input video V_i.

It finally produces an output video with frames containing pixels given by:

$$p'_i = \begin{cases} c_{\min O} & \text{when } O \neq \emptyset \\ \omega & \text{otherwise} \end{cases}$$

Q2(c) is verified using semantic validation, where each detected object is mapped back to an actual object in the scene geometry that produced the input video, and a reference bounding box is generated for the object. The maximum Jaccard distance between the VDBMS-generated and reference boxes must not exceed ϵ. In the prototype version of Visual Road we describe in Section 5, we have adopted the PASCAL VOC [16] threshold of $\epsilon = 0.5$.

Query Q2(d) performs background masking on each input video by applying a mean filter [3] to each video frame. Background masking is useful for removing static, unchanging regions of a frame (e.g., sidewalks and buildings) and leaving the dynamic "foreground" areas untouched. For each window of m video frames $f_j, \ldots, f_{j+m}$ in an input video, a VDBMS should compute a background reference frame $b_j = \frac{1}{m} \sum_{k \in [i..i+m]} f_k$. Next, for each pixel p_v in frame f_j and p_b in background reference frame b_j, the VDBMS should output a black pixel ω when their difference is below the threshold $\lvert\frac{p_v - p_b}{p_v}\rvert < \epsilon$ and p_v otherwise.

Finally, query Q3 performs an operation on individual regions of each frame in an input video. For example, an application might deliver less-important regions at lower bitrates (see Q10) or blur regions of a video frame that contain faces or other sensitive information. Q3 performs the former operation by segmenting input frames into regions of size $(\Delta x, \Delta y)$. Each resulting region u_i is re-encoded at a bitrate given by b_i. The resulting regions should then be recombined.

4.1.2 *Computer Vision.*

Interpolation & Resampling (Q4, Q5). Computer vision algorithms and machine learning models frequently require an input image sampled at a particular resolution. These queries test a VDBMS's ability to perform this sampling by asking it to perform interpolation and resampling operations on input videos. First, query Q4 increases each input video's resolution to $(\alpha R_x, \beta R_y)$ using bilinear interpolation. Query Q5 performs the inverse operation: given an input video, the VDBMS downsamples each frame to a lower resolution $(\frac{R_x}{\alpha}, \frac{R_y}{\beta})$.

Union (Q6). Modern video applications frequently require combining two or more data streams. For example, an augmented reality application might overlay advertising or informational text on a user's display. Queries Q6(a) and Q6(b) test a VDBMS's ability to perform these operations by merging and combining data stored in various formats.

In particular, query Q6(a) merges an input video V_i with a *bounding box video* $B = Q_{2c}(V_i)$ by performing an outer join on the corresponding pixels within each video. The bounding box video is generated offline by the VCD by applying the reference implementation of query Q2(c) to the associated input video. For each pair of corresponding pixels ($p_j \in V_i, b_j \in B_i$), a VDBMS produces an output pixel using a ω-*coalesce* projection given by:

$$p'_j = \begin{cases} b_j & \text{when } b_j \neq \omega \\ p_j & \text{otherwise} \end{cases} \quad (1)$$

The VCD exposes B in two formats: as an encoded video and as a serialized sequence of bounding box class identifiers and coordinates. VDBMSs may consume either format when executing the query. As in Q2(c), the VCD uses the black sentinel color ω to represent null pixels in the encoded variant.

Query Q6(b) overlays a set of text annotations C_i onto an input video. Like query Q6(a), this query uses Equation 1 to generate output pixel colors. However, here the input C_i is a WebVTT [43] file embedded as a metadata track within the input video's container. The VCD randomly generates the WebVTT file and randomly varies position and non-overlapping duration of each annotation. Benchmarking VDBMSs may render the annotations using any font, and need only support the `line` and `position` cue settings.

4.2 Composite Benchmarks

This section describes more complex, real-world workloads that we call *composite benchmarks*. Each composite benchmark leverages one or more of the microbenchmarks introduced in the previous section. Composite benchmarks are drawn from recent examples and applications in the computer vision and machine learning literature (see Section 7).

4.2.1 *Computer Vision.*

Object Detection (Q7). This query leverages Visual City cameras to identify instances of a given object class $o \in O$ (e.g., pedestrians or vehicles). To draw attention to identified objects, it also removes extraneous "background" portions of each video frame that do not contain visual information about the class and persists or streams the results.

At a high level, a VDBMS implementing this query first applies the classification query Q2(c) to every input video. Next, for each object type, it overlays the resulting bounding boxes onto the input videos using query Q6(a). Finally, it refines the results by performing background removal as defined in Q2(d). Figure 3 illustrates this process applied to a single video frame.

(a) Input video frame (b) Output video frame

Figure 3: Sample input and output for one frame of the object detection query (Q7).

Table 6: Object detection query (Q7).

Input	Input videos $\{V_1, ..., V_n\}$
	Object detection function $A(V, O)$
	Object classes $O = \{o_1, ..., o_m\}$
Output	Videos $\{V_1^{o_1}, ..., V_n^{o_m}\}$ where
	$V_j^{o_i} = Q_{2d}(Q_{6a}(V_i, Q_{2c}(V_j, A, \{o_i\})))$

As formalized in Table 6, the inputs to the object detection query are the videos generated by the traffic cameras scattered throughout Visual City. A VDBMS may report results using additional object classes or detection algorithms, so long as it also includes results for those defined in Table 3.

Vehicle Tracking (Q8). Query 8 simulates the tracking of vehicle sightings throughout Visual City. Each automobile in Visual City has a unique front-facing license plate containing six random alphanumeric digits.

A vehicle sighting instance is defined by the period in which it is identifiable by one or more traffic cameras. Initially, a vehicle *enters* a traffic camera's field of view when its license plate is unobscured relative to that camera. It *exits* the traffic camera's field of view when one or both of these conditions is no longer met. The video frames occurring between a vehicle entering and exiting a camera is a *vehicle tracking segment* (VTS).

The VCD simulates searching for vehicles by issuing *vehicle tracking queries* to the VDMBS. The input to this query is the license plate of a random vehicle. As illustrated in Figure 4, the output is a *tracking video* of temporally-ordered (by entry time), concatenated VTSs for the vehicle associated with that license plate.

This query is formalized in Table 7 as a recurrence. Its output is defined by repeated application of Q2(c). Each application uses a license plate recognition function $\mathcal{L}$ to identify the next VTS_i in the input video. Query Q1 is used to select the temporal range $[t_i, t_{i+1}]$ and the output is appended to the previous iteration until a fixpoint is reached.

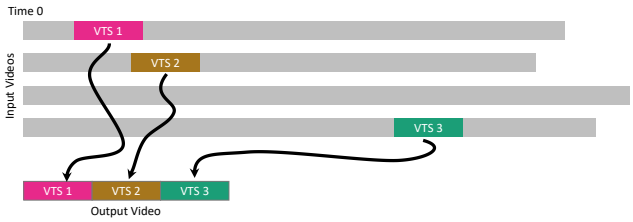

Figure 4: Illustration of the vehicle tracking query (Q8) on a Visual Road dataset (scale = 1) that contains three vehicle tracking segments (VTSs). Each VTS is temporally-ordered, concatenated, and output.

Table 7: Vehicle tracking query (Q8)

Input	Traffic camera videos $\{V_1, ..., V_n\}$ License plate $l = (l_1, ..., l_6)$ License plate recognition function $\mathcal{L}$ (OpenALPR)
Output	Video $V_{\text{out}} = VTS_1 \oplus \cdots \oplus VTS_n$ where $t_i = \sum_{j \in [1..i-1]} Duration(VTS_j)$ $VTS_i = Q_1(Q_{6a}(V_i, Q_{2c}(V_i, \mathcal{L}, \{l\}))$ $(t_i, t_{i+1})))$

4.2.2 Virtual Reality.

Virtual reality (VR) video is an important, emerging subclass of video data. Panoramic VR videos (a.k.a 360° videos) are one popular member of this subclass. Visual Road includes two benchmark queries that target the VR 360° data format. We include these queries because VR video operations exercise sophisticated features possible only in the most recent VDBMSs, and evaluating their performance at scale is an important differentiating factor between such systems. Operations on VR videos are also useful to test a VDBMS's ability to use higher resolutions than typically seen in ordinary 2D video.

The following queries are more open-ended than the previous benchmark categories, allowing an implementing VDBMS additional freedom to optimize their execution.

Panoramic Stitching (Q9). Modern panoramic cameras produce video panoramas by "stitching" together two or more ordinary 2D videos into a 360° video. To take advantage of modern video compression, the spherical video is reprojected onto a plane and encoded as if it were an ordinary 2D video. Query Q9 requires that a VDBMS perform this process by stitching video data from the panoramic cameras scattered throughout Visual City. Recall from Section 3 that each panoramic camera is composed of four ordinary 2D cameras with a 120° field of view. A VDBMS implementing Q9 should accept the video streams from the constituent 2D cameras, execute a function to convert the four images into a single 360° video, and output it. This process should be repeated for every panoramic camera in Visual City.

Table 8: Tile-Based Streaming (Q10).

Input	360° videos $\{U_1, ..., U_n\}, U_i = Q_9(V_i)$ Bitrates $B = (b_1, ..., b_9), b_i \in \{b_h, b_l\}$ Client resolution $R_c = \{r_1, ..., r_n\}$
Output	Videos $\{V'_1, ..., V'_n\}$ where $V'_i = Q_5(Q_3(V_i, j \rightarrow b_j), r_i)$

A VDBMS is free to implement the conversion in any manner that is most efficient, with the constraints that (i) the resulting 360° videos should be equirectangularly projected [48] and (ii) the result should be moderately similar (i.e., within 30 dB PSNR) of the reference implementation.

Tile-Based Encoding (Q10). Recent research has suggested that streaming "unimportant" areas of a 360° video in lower resolution may yield substantial bandwidth savings [21, 24, 59]. Additional savings may be achieved by reducing the resolution of a VR video to match the resolution of the VR headset or viewing device. This query, formalized in Table 8, measures a VDBMS's ability to use both techniques to reduce bandwidth costs. To execute this query, a VDBMS should use Q3 to decompose each video frame into nine equal-sized "tiles" and encode high-importance tiles at a high-quality bitrate b_h and the remaining tiles at a low-quality bitrate b_l. The VDBMS should also use Q5 to downsample the video to a lower resolution that matches the viewing device. For simplicity, we treat these parameters as global values that are applied over the entire duration of the input 360° video.

5 IMPLEMENTATION

We implement the video generators for Visual Road 1.0 by adapting CARLA 0.84 [13], an open-source simulator designed for autonomous driving research. CARLA itself is designed as a "plugin" for the Unreal Engine 4.18, a commercial gaming engine that provides physics, simulation, and other graphics-oriented features. CARLA includes resources, textures, and geometry, which form the basis of the tiles used in Visual Road. It also exposes a configuration-driven API that facilitates camera placement, rendering, and other convenience functionality. We modify CARLA to support efficient video encoding, camera rendering at varying resolutions and frame rates, and multiple tiles and configurations. All artifacts are developed using C++.

Version 1.0 of Visual Road contains a tile pool consisting of 72 tiles. Each tile is constructed from one of two maps (TOWN01 and TOWN02), both drawn from the set of CARLA resources. Each is also associated with one of twelve different weather configurations and one of three different vehicle and pedestrian densities (e.g., a "rush hour" tile contains 120 vehicles and 512 pedestrians). Each tile is configured with 4 traffic cameras and 1 panoramic camera, both capturing at 30 frames per second.

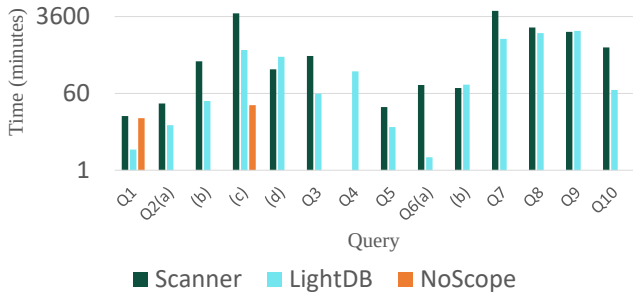

Figure 5: Visual Road log-scale performance by query with scale factor $L = 4$ at 1K and 60 minutes.

We also develop a Visual Road reference implementation for use in verifying benchmark results using PSNR comparisons. The reference implementation was written in C++ and depends on FFmpeg [6] for video-related operations. For semantic verification, the reference implementation interacts with the Unreal Engine to generate metadata relating to objects in a camera's frame of view.

To generate the videos that serve as benchmark, the VCG (see Section 3) produces a sequence of *video frames* for each camera in the city. These frames are periodic temporal samples of visual data. Each element of a frame at (x, y) is a pixel containing a color in a given color space (e.g., RGB, YUV). Frames are physically sequenced using a constant temporal period (i.e., a frame rate). Each video has a duration D and constant frame *resolution* $R = (R_x, R_y)$. Finally, video codecs compress each frame to produce an encoded video, and each encoded input video is separately muxed using the MP4 container format [29].

The current version of Visual Road includes support for H264 [57] and HEVC [51], and each query result must be encoded using either of these codecs. Visual Road also currently supports frame rates in the range of 15-90 frames per second (FPS) and resolutions at 1K (960 × 540), 2K (1920 × 1080), and 4K (3840 × 2160). However, we anticipate that future versions will extend support to additional codecs, containers, frame rates, and resolutions.

The VCG supports single-node and distributed modes of input video generation. In distributed mode, the VCG uses the Amazon Elastic Compute Cloud (EC2) to launch multiple instances of the Unreal Engine in parallel. Each node independently configures the Visual Road environment, launches an Unreal Engine instance, simulates the tile for which it is responsible, and collects video output.

6 EVALUATION

We experimentally evaluate Visual Road in three ways. First, in Section 6.1, we demonstrate that Visual Road produces performance results for VDBMSs similar to real-world datasets and better than alternative synthetic approaches. Next, in Section 6.2, we apply the benchmark to three recent open-source VDBMSs and contrast the results. For these experiments we show out-of-the-box performance numbers for all of the experiments. Better results could certainly be achieved for each system with appropriate tuning. Our goal is to evaluate the benchmark and not the systems. Next, in Section 6.3.1 we evaluate the quality of the video generated by Visual Road, and in Section 6.4 we evaluate the performance differences between write and streaming execution modes. Finally, in Section 6.3.2 we evaluate the scalability of Visual Road when generating large corpora.

Experimental configuration. Except where stated otherwise, we perform experiments using a hardware configuration consisting of a single physical machine running Ubuntu 16.04 and containing an Intel 3.4 Ghz i7-6800K processor with 6 cores and 32 GB RAM. It is equipped with a 256 GB SSD drive and an Nvidia Quadro P5000 GPU.

Benchmarked VDBMSs. To show wide applicability, we execute Visual Road on three recent, open-source VDBMSs: Scanner [44], LightDB [20], and NoScope [33]. These VDBMSs cover a variety of target use-cases, respectively including processing at scale, virtual reality video, and specialized application of deep learning models.

Scanner is an open-source VDBMS that offers efficient distributed video processing at scale. We deploy Scanner using its most recently-published Docker container, which was built using CUDA 8.0 [39], OpenCV 3.2 [42], and FFmpeg 3.3.1 [6]. Scanner lacks support for video cropping (Q1), captioning (Q6(b)), and license plate recognition (Q8), so we add these features as custom C++ operators (respectively) using a modified resize operator, the `libwebvtt` [38], and `libopenalpr` [41]. We also make minor modifications to Scanner's grayscale and resizing kernels so that queries Q2(a) and Q4-5 can be expressed.

LightDB is a VDBMS specialized for virtual reality video workloads. We deploy the most recent prototype of this VDBMS, which depends on CUDA 8.0 and FFmpeg 2.8. LightDB exposes operations that accept angles rather than pixel offsets, and so we adapt each benchmark query by manually mapping between the two coordinate systems. LightDB also lacks support for Q6(b) captioning and Q8 license plate recognition, and so we use LightDB's C++ "plugin" mechanism—again using `libwebvtt` and `libopenalpr`—to add support for these features.

Finally, NoScope is a specialized VDBMS that improves the performance of applying deep learning models to video at scale. We deploy the most recent prototype of NoScope, which relies on TensorFlow 0.12, CUDA 8.0, and cuDNN 5.1. Because NoScope is specialized for deep learning and does not expose support for arbitrary queries or a mechanism for extensibility, we are only able to express queries Q1 and Q2(c) using this system.

Table 9: Visual Road ability to accurately measure VDBMS performance compared with real videos. Values show total runtime in minutes and speedup relative to the UA-DETRAC baseline for LightDB and Scanner. Red cells indicate a result where the relative performance between systems differs from the baseline, while Yellow cells show performance discrepancies of an order of magnitude or more relative to the baseline.

Query	UA-DETRAC		Visual Road		Duplicates		Random	
	LightDB	Scanner	LightDB	Scanner	LightDB	Scanner	LightDB	Scanner
Q1	1	2	1 (0.9×)	2 (0.8×)	1 (0.7×)	2 (1.0×)	3 (4×)	61 (26×)
Q2(a)	1	4	1 (0.7×)	3 (0.8×)	1 (0.8×)	4 (0.9×)	5 (4×)	4 (1×)
Q2(b)	8	36	5 (0.6×)	25 (0.7×)	1 (0.2×)	31 (0.9×)	9 (1.1×)	43 (1.2×)
Q2(c)	25	472	23 (0.9×)	360 (0.8×)	3 (0.1×)	432 (0.9×)	25 (1×)	451 (1×)
Q2(d)	32	18	30 (0.9×)	19 (1.0×)	6 (0.2×)	19 (1.1×)	118 (4×)	57 (3×)
Q3	13	45	9 (0.7×)	43 (0.9×)	1 (0.1×)	46 (1.0×)	158 (13×)	313 (7×)
Q4	26	N/A	25 (0.9×)	N/A	16 (0.6×)	N/A	103 (4×)	N/A
Q5	1	4	1 (0.8×)	3 (0.6×)	1 (0.4×)	4 (0.9×)	24 (19×)	13 (3×)
Q6(a)	2	14	2 (0.9×)	13 (0.9×)	1 (0.4×)	15 (1.1×)	29 (16×)	19 (1.4×)
Q6(b)	12	11	11 (0.9×)	8 (0.7×)	2 (0.2×)	11 (0.9×)	53 (5×)	66 (6×)

We execute all queries in this section using VCD's offline mode, since neither Scanner nor NoScope support operating on live-streaming video data. Except where stated otherwise, for all systems we use default settings and did not attempt to optimize batch size or leverage other optimization strategies.

6.1 Dataset Validation

In this section, we evaluate whether Visual Road's synthetically generated data yields performance numbers similar to using real videos. We also evaluate whether other types of synthetic inputs could work as well as Visual Road to test a VDBMS. Overall, we find that Visual Road-generated input videos produce runtimes similar to using real-world, manually-annotated data, whereas other synthetic datasets may yield misleading or incorrect results.

As a real-world baseline, we use the UA-DETRAC [56] video dataset. UA-DETRAC is a manually-annotated corpus composed of recorded traffic camera videos of various durations. Our experiments in this section utilize the training subset, which consists of 60 sequences of 1K video recorded at 25 FPS. The data are provided as 83,791 images, which we H264-encode to produce approximately two hours of video.

We next use Visual Road 1.0 to create input videos that match the UA-DETRAC configuration. We execute the VCG with scale factor $L = 16$ at 1K resolution to produce 64 traffic camera videos. From these, we randomly select 60 videos and re-encode each to 25 FPS. We finally truncate each video so its duration matches a corresponding video in the UA-DETRAC dataset.

In addition to comparing with the Visual Road-generated input videos, we also construct two alternative synthetic datasets as follows:

Duplicate videos. A user might test a VDBMS by reproducing one or more manually-annotated videos to create a larger synthetic corpus. To simulate this process, we select the longest UA-DETRAC video ("MVI_40172") and replicate it 60 times. We then truncate each replicated video to match the duration of a corresponding video in the UA-DETRAC dataset.

Random videos. Alternatively, a user might use randomly-generated video to evaluate VDBMS performance. To simulate this approach, we generate a fully-synthetic video corpus consisting of random noise. As in the previous dataset, we generate 60 videos matched in duration to UA-DETRAC.

We execute the microbenchmark queries on the Scanner and LightDB VDBMSs using each of the datasets described above. Because NoScope is only able to execute two of the queries, we omit it from this experiment. We were not able to execute Q4 on Scanner for reasons we describe in Section 6.2.

Table 9 shows the performance results for each VDBMS, query, and dataset. For each of the queries, VDBMS performance for the Visual Road input is similar to the UA-DETRAC input. In no cases does the Visual Road dataset lead to a result that disagrees with the UA-DETRAC counterpart, meaning that the benchmark correctly identifies the faster system for each query. In general, performance for each query closely tracks the baseline.

The duplicate and random datasets do not consistently agree with the UA-DETRAC performance results. For these datasets, at least one query produces a result that disagrees with the baseline (i.e., where system X performs faster than system Y on UA-DETRAC but worse on the synthetic input) and could lead a user to draw an incorrect inference about the performance of a system when using real-world video.

Equally problematic are the cases where the performance differences between systems differ by more than an order of magnitude compared to the baseline dataset. We have highlighted discrepancies of this magnitude on Table 9. Such a difference occurs for more than one query in both the

duplicate and random datasets, and could lead a user to draw an incorrect conclusion about the relative performance differences between VDBMSs when using one of these synthetic datasets.

Overall, system performance on Visual Road data is similar to the real videos with the important advantage that Visual Road data is synthetically generated and videos can thus be scaled and parameterized as needed.

6.2 System Comparison

In this section, we apply the benchmark to the comparison VDBMSs at various scale factors and show that Visual Road is a useful benchmark for comparing performance between systems. The time to generate a Visual Road dataset need only be incurred once since a given configuration determines the resulting videos.

Our first experiment gives a high-level overview of VDBMS performance. In this experiment, we hold constant the scale factor ($L = 4$), resolution (1K), and duration (1 hour). We use this configuration and execute applicable benchmark queries on each VDBMS.

Figure 5 shows the log-scale total runtime for each system and query combination. NoScope shows excellent performance on Q2(c)—which closely matches the workloads it was designed to execute—but its highly specialized implementation doesn't support most of the other benchmark queries. Scanner and LightDB show similar performance on Q1, Q6(b), and Q7-Q10.

We next vary the scale factor L while holding other parameters constant at their previous values. To accomplish this, we used the VCG to generate a series of one-hour datasets at 1K resolution with increasing the size of the simulated city. We then execute each query on a VDBMS and measure the total runtime until completion. As we discussed previously, the NoScope system only supports Q1 and Q2(c) and so we show results only for these queries.

Figure 6 shows detailed VDBMS performance for each benchmark query. At small scale factors, no single system dominates across all queries. As the scale factor increases, however, Scanner often falls behind the other comparison systems. This drop-off appears to be due to memory thrashing as more video data are introduced. Scanner also suffers from a poorly-performing resize kernel (Q1) and its use of the Caffe [30] deep learning framework to execute the Q2(c) YOLO neutral network.

LightDB performs well across many queries, but suffers from a CPU-only implementation of the captioning query. As expected, NoScope excels at efficiently applying the YOLO neural network in query Q2(c).

Both Scanner and LightDB have memory-related issues when executing Q4. When we execute this query on Scanner, it quickly allocates all available memory and thereafter fails to make progress. This occurs even when we attempt to execute Q4 on Scanner with one input video or with a custom, Python-based implementation of the resize operator.

LightDB exhibits similar issues when attempting to subquery (Q3) or resize (Q4) more than 40 videos, after which it fails due to lack of GPU memory. We work around this by issuing these queries in two batches — one with the first 40 videos, and a second with the rest.

We also observe that the composite and VR benchmark queries took far longer for both systems than did the microbenchmark queries (with the exception of Q2(c), which requires executing an expensive convolutional neural network). This supports that Visual Road is effectively targeting a wide range of workload complexities.

Our final comparison shows the lines of code (LOC) required to execute each query on a VDBMS. To calculate LOC, we construct a file containing the minimal code required to execute each query, auto-format it, and count the number of non-empty lines. Scanner and NoScope expose Python bindings and we use AUTOPEP for formatting; we similarly use CLANG-FORMAT for LightDB's C++ API. We separately count implementation for queries that required additional logic (e.g., LightDB's text caption plugin for Q6(b)) using the same approach.

Figure 7 shows the resulting counts. Here, Scanner and LightDB have similar LOC counts for many queries. The same is true for supporting extension implementation, primarily because both are written in C++. Because NoScope narrowly targets only a single query, invoking it requires only a few lines of Python code.

Overall, Visual Road effectively shows that NoScope is an excellent, highly specialized engine while Scanner and LightDB are more general purpose. It also exposes the performance advantages and limitations of each system on the different query types.

6.3 Video Quality & Generation Time

6.3.1 Quality of Video. In this section we examine the quality of video produced by Visual Road and how similar it is to real video. Again, our goal is to provide evidence that videos are of good enough quality to be used to evaluate query execution time.

To evaluate this aspect, we use the YOLOv2 [46] model to identify automobiles (i.e., cars and vans) in both synthetic Visual Road and real UA-DETRAC video. This model comes pretrained on the COCO training/validation dataset [34]. Each test set contains 1920 randomly-selected frames.

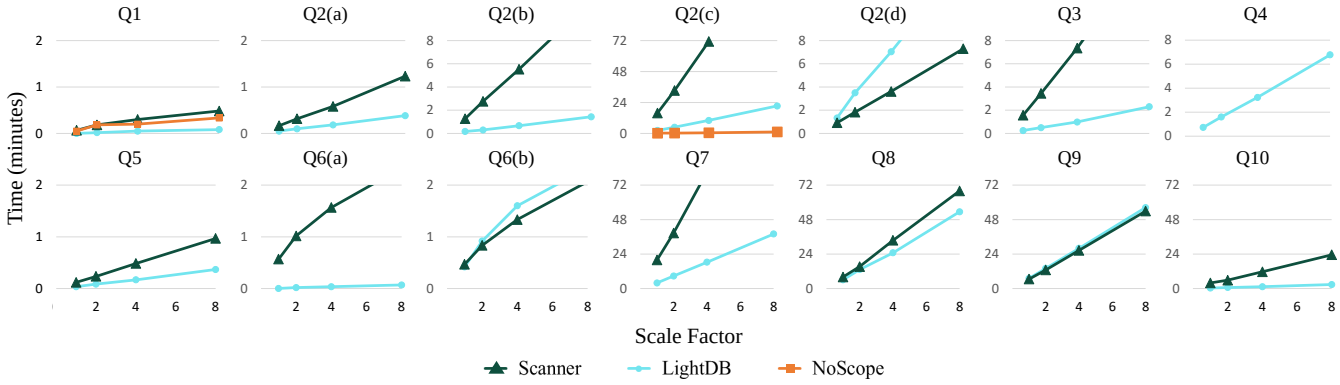

Figure 6: VDBMS performance showing, at various scale factors, the total time to execute the benchmark queries.

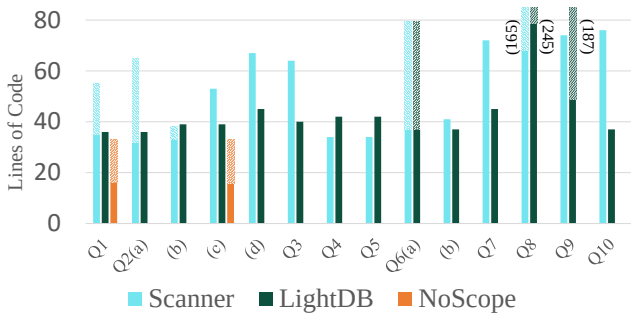

Figure 7: Lines of code (LOC) required to execute each supported benchmark query. Solid bars show LOCs to implement query and hashed bars for supporting extension code. LOCs over 80 shown in parenthesis.

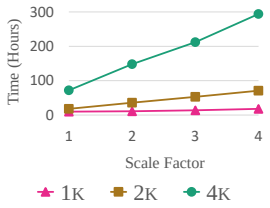

Figure 8: Performance by scale/resolution (4 nodes, 60 minutes)

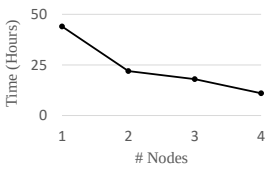

Figure 9: Performance by #nodes (scale 2, 1k, 60 minutes).

The average precisions (APs) at 50% IoU for the Visual Road and UA-DETRAC datasets were respectively 72 and 75%. This is similar to results reported by Redmon & Farhadi (AP = 77% [46]) for this model on the "car" category of another benchmark dataset [16]. This suggests that the semantic structure of the synthetic Visual Road video is similar to that of real video and supports its use for evaluating the query execution time of a VDBMS at scale. However, these results notwithstanding, we would like to again emphasize that Visual Road is not intended to train machine learning models or evaluate a VDBMS in terms of prediction accuracy.

6.3.2 Generator Performance. We next explore the performance of the Visual Road Generator (VCG) when creating large video datasets. Figure 8 shows the total time to generate a one-hour dataset with increasing scale factor and at three resolutions: 1K, 2K, and 4K. For this experiment, we executed the VCG on a single node using the hardware configuration described previously.

These results show an approximately linear increase in single-node generation time as the scale factor increases. This result is intuitive, since (i) the number of cameras is a linear function of scale factor, (ii) at a constant resolution the total number of generated pixels increases linearly with number of cameras, and (ii) the underlying scene geometry must be recalculated on a per-camera basis, precluding opportunities to render in parallel. The 4K generation time increases more rapidly due to a software limitation related to the number of cameras that can be simultaneously instantiated; we plan to further optimize this in the future.

We next evaluate the performance of the VCG in distributed mode when generating video in parallel using multiple nodes. We hold constant scale factor ($L = 2$), resolution (1K), and duration (1 hour), and vary the number of nodes used to execute the VCG. For this experiment, we use `p3.2xlarge` nodes on the Amazon Elastic Compute Cloud (EC2) to generate video in parallel. Each instance has one Nvidia V100 GPU, 8 logical cores, and 61GiB of RAM.

Figure 9 shows the time required by the VCG to generate a dataset with the above configuration and given number of nodes. Because dataset generation does not require coordination between cameras, we see an expected linear decrease in generation time as we increase the number of nodes available for processing.

6.4 Write & Discard Modes

Our final set of experiments evaluate the performance differences between benchmark execution in write and streaming modes (see Section 3.2). To do so, we executed the benchmark on the Scanner and LightDB systems in each execution mode. To support streaming mode on Scanner, we modified each query to send results to the null device. We used LightDB's sink operator for this operation.

For each query, we found that the performance difference between the two modes was less than 2.5%. This difference is in part due to pipelineing and also because disk IO is inexpensive relative to video compression.

7 RELATED WORK

The database community has a long history of standardizing on various benchmarks that target a wide range of data management applications. These applications include longstanding topics such as OLTP [53], OLAP [44], and streaming [2]. They also cover more modern areas such as the Internet of things [4], block chains [12], social networks [15], and big data analytics [52]. Visual Road complements these benchmarks by extending robust support for performance evaluation of video processing at scale and motivates future work in querying functionality and performance.

To date, we are aware of no video performance benchmark that scales to an unlimited duration or resolution and does not require manual annotation, despite the fact that a number of recent VDBMSs have emerged to support a wide range of modern applications. We target most of these applications in this benchmark (e.g., license plate recognition [35, 60] (Q8); background subtraction/masking [35] (Q2(d), Q6); general object detection [20, 25, 31, 32, 44, 60] (Q2(c), Q7); decode performance [20, 44], stitching [26, 44] (Q9); up/downsampling [26] (Q4,Q5,Q10); user-defined transformations [1] (Q2(a-d)); tile-based encoding [20, 21, 24, 59] (Q10)). These VDBMSs differ from previous-generation VDBMSs in that they support applications beyond simple content search and information retrieval. Representative systems include those shown in Table 1. Other work targets specific aspects of video data management such as predicate push-down [36]. Each system benefits from Visual Road, which enables objective comparison of features and performance.

Several recently-released video-oriented simulation frameworks target specific aspects of AI model training. These include CARLA [13]—on which the Visual Road prototype is built—along with others such as AirSim [50] and DeepDrive [11]. Other specialized systems target military (e.g., UTSAF [45]) or medical applications (e.g., [8]). While these frameworks all use modern simulation and visualization software (e.g., Unreal), they are not designed to produce large amounts of heterogeneous video, nor do they come with queries useful for evaluating VDBMS performance.

A number of video-oriented datasets and benchmarks have also emerged that target various aspects of machine learning. These include UA-DETRAC [56], BDD100K [58], ApolloScape [27], WebCamT [61], and ActivityNet [22]. While these might be used to evaluate VDBMS performance, they are of fixed, modest size and must be laboriously annotated.

Generalizability and transferability of results is a significant challenge to applications that leverage synthetic data for use in real-world applications. Prior work in several areas have examined this issue. For example, in their survey of robot simulators, Craighead et al. argued that contemporaneous simulation software had high physical fidelity [9]. In a subsequent survey on UAV and robot simulators, Cook et al. drew similar conclusions in oceanographic robotics with respect to the physics engines [7]. In the computer vision domain, researchers have evaluated the transferability of models learned on synthetic data to real-world applications. Previous approaches have used various transferability metrics (e.g., precision/recall [19], multi-object tracking accuracy [18], collision-free percentage [47], average accuracy [10], ROC curve [37], observed similarity [50]). Some previous work has demonstrated that synthetic data leads to superior models when data is limited or of low variety [19]. Visual Road evaluates transferability using an approach similar to Hattori et al. [19].

8 CONCLUSION & FUTURE WORK

We presented Visual Road, a benchmark for video data management systems (VDBMSs). Visual Road comes with a data generator that produces an unlimited amount of synthetic video generated by simulating an active metropolitan area, along with a suite of queries that evaluate VDBMS performance. Our results show that video generated using Visual Road closely matches real-world, manually-annotated video corpora and allows VDBMSs to be evaluated at any scale. We used an initial implementation of the Visual Road benchmark to evaluate the performance of several modern VDBMSs and show that it is a useful tool for capturing meaningful performance comparisons between systems. As visualization and simulation tools evolve, future versions of Visual Road will automatically fuse tiles, track objects across tiles, and support increasingly complex procedurally-generated tiles.

Acknowledgments. This work is supported by the NSF through grants CCF-1703051, IIS-1546083, CCF-1518703, and CNS-1563788; DARPA award FA8750-16-2-0032; DOE award DE-SC0016260; a Google Faculty Research Award; an award from the University of Washington Reality Lab; gifts from the Intel Science and Technology Center for Big Data, Intel Corporation, Adobe, Amazon, Facebook, Huawei, and Google; and by CRISP, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

REFERENCES

[1] Lixiang Ao, Liz Izhikevich, Geoffrey M. Voelker, and George Porter. 2018. Sprocket: A Serverless Video Processing Framework. In *SoCC*. 263–274.

[2] Arvind Arasu, Mitch Cherniack, Eduardo F. Galvez, David Maier, Anurag Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. 2004. Linear Road: A Stream Data Management Benchmark. In *PVLDB*. 480–491.

[3] Gonzalo R. Arce. 2004. *Nonlinear Signal Processing - A Statistical Approach*. Wiley.

[4] Martin F. Arlitt, Manish Marwah, Gowtham Bellala, Amip Shah, Jeff Healey, and Ben Vandiver. 2015. IoTAbench: an Internet of Things Analytics Benchmark. In *ICPE*. 133–144.

[5] Mohammadamin Barekatain, Miquel Martí, Hsueh-Fu Shih, Samuel Murray, Kotaro Nakayama, Yutaka Matsuo, and Helmut Prendinger. 2017. Okutama-Action: An Aerial View Video Dataset for Concurrent Human Action Detection. In *CVPR*. 2153–2160.

[6] Fabrice Bellard. 2018. FFmpeg. https://ffmpeg.org.

[7] Daniel Cook, Andrew Vardy, and Ron Lewis. 2014. A survey of AUV and robot simulators for multi-vehicle operations. In *AUV*. 1–8.

[8] Brent Cowan, Hamed Sabri, Bill Kapralos, Sayra Cristancho, Fuad Moussa, and Adam Dubrowski. 2011. SCETF: Serious game surgical cognitive education and training framework. In *IGIC*. 130–133.

[9] Jeff Craighead, Robin R. Murphy, Jenny Burke, and Brian F. Goldiez. 2007. A Survey of Commercial & Open Source Unmanned Vehicle Simulators. In *ICRA*. 852–857.

[10] César Roberto de Souza, Adrien Gaidon, Yohann Cabon, and Antonio Manuel López Peña. 2017. Procedural Generation of Videos to Train Deep Action Recognition Networks. In *CVPR*. 2594–2604.

[11] DeepDrive 2018. DeepDrive: Self-Driving AI. https://deepdrive.io.

[12] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. 2017. BLOCKBENCH: A Framework for Analyzing Private Blockchains. In *SIGMOD*. 1085–1100.

[13] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *CoRL*. 1–16.

[14] Epic Games. 2019. Unreal Engine 4. https://www.unrealengine.com.

[15] Orri Erling, Alex Averbuch, Josep-Lluís Larriba-Pey, Hassan Chafi, Andrey Gubichev, Arnau Prat-Pérez, Minh-Duc Pham, and Peter A. Boncz. 2015. The LDBC Social Network Benchmark: Interactive Workload. In *SIGMOD*. 619–630.

[16] Mark Everingham, Luc J. Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. 2010. The Pascal Visual Object Classes (VOC) Challenge. *IJCV* 88, 2 (2010), 303–338.

[17] Robert B Fisher. 2004. The PETS04 surveillance ground-truth data sets. In *PETS*. 1–5.

[18] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. 2016. VirtualWorlds as Proxy for Multi-object Tracking Analysis. In *CVPR*. 4340–4349.

[19] Hironori Hattori, Vishnu Naresh Boddeti, Kris M. Kitani, and Takeo Kanade. 2015. Learning scene-specific pedestrian detectors without real data. In *CVPR*. 3819–3827.

[20] Brandon Haynes, Amrita Mazumdar, Armin Alaghi, Magdalena Balazinska, Luis Ceze, and Alvin Cheung. 2018. LightDB: A DBMS for Virtual Reality Video. *PVLDB* 11, 10 (2018), 1192–1205.

[21] Brandon Haynes, Artem Minyaylov, Magdalena Balazinska, Luis Ceze, and Alvin Cheung. 2017. VisualCloud Demonstration: A DBMS for Virtual Reality. In *SIGMOD*. 1615–1618.

[22] Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. 2015. ActivityNet: A large-scale video benchmark for human activity understanding. In *CVPR*. 961–970.

[23] Alain Horé and Djemel Ziou. 2010. Image Quality Metrics: PSNR vs. SSIM. In *ICPR*. 2366–2369.

[24] Mohammad Hosseini and Viswanathan Swaminathan. 2016. Adaptive 360 VR Video Streaming Based on MPEG-DASH SRD. In *ISM*. 407–408.

[25] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodík, 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 *OSDI*. 269–286.

[26] Qi Huang, Petchean Ang, Peter Knowles, Tomasz Nykiel, Iaroslav Tverdokhlib, Amit Yajurvedi, Paul Dapolito IV, Xifan Yan, Maxim Bykov, Chuen Liang, Mohit Talwar, Abhishek Mathur, Sachin Kulkarni, Matthew Burke, and Wyatt Lloyd. 2017. SVE: Distributed Video Processing at Facebook Scale. In *SOSP*. 87–103.

[27] Xinyu Huang, Xinjing Cheng, Qichuan Geng, Binbin Cao, Dingfu Zhou, Peng Wang, Yuanqing Lin, and Ruigang Yang. 2018. The ApolloScape Dataset for Autonomous Driving. *CoRR* abs/1803.06184 (2018).

[28] Sinisa Ilic, Mile Petrovic, Branimir Jaksic, Petar Spalevic, Ljubomir Lazic, and Mirko Milosevic. 2013. Experimental analysis of picture quality after compression by different methods. *Przeglad Elektrotechniczny* (2013), 0033–2097.

[29] International Organization for Standardization. 2003. *Coding of audio-visual objects – Part 14: MP4 file format*. Standard ISO/IEC 14496-14:2003.

[30] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. In *ACMMM*. 675–678.

[31] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodík, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: scalable adaptation of video analytics. In *SIGCOMM*. 253–266.

[32] Daniel Kang, Peter Bailis, and Matei Zaharia. 2018. BlazeIt: Fast Exploratory Video Queries using Neural Networks. *CoRR* abs/1805.01046 (2018).

[33] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Deep CNN-Based Queries over Video Streams at Scale. *PVLDB* 10, 11 (2017), 1586–1597.

[34] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. In *ECCV*. 740–755.

[35] Yao Lu, Aakanksha Chowdhery, and Srikanth Kandula. 2016. Optasia: A Relational Platform for Efficient Large-Scale Video Analytics. In *SoCC*. 57–70.

[36] Yao Lu, Srikanth Kandula, and Surajit Chaudhuri. 2018. Interactive Demonstration of Probabilistic Predicates. In *SIGMOD*. 1669–1672.

[37] Javier Marín, David Vázquez, David Gerónimo, and Antonio M. López. 2010. Learning appearance in virtual scenarios for pedestrian detection. In *CVPR*. 137–144.

[38] Mozilla Foundation and Contributors 2018. WebVTT parsing library. https://github.com/hihihippp/webvtt-3.

[39] NVIDIA Corporation. 2007. *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. NVIDIA Corporation.

[40] Sangmin Oh, Anthony Hoogs, A. G. Amitha Perera, Naresh P. Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, J. K. Aggarwal, Hyungtae Lee, Larry S. Davis, Eran Swears, Xiaoyang Wang, Qiang Ji, Kishore K. Reddy, Mubarak Shah, Carl Vondrick, Hamed Pirsiavash, Deva Ramanan, Jenny Yuen, Antonio Torralba, Bi Song, Anesco Fong, Amit K. Roy-Chowdhury, and Mita Desai. 2011. A large-scale benchmark dataset for event recognition in surveillance video. In *CVPR*. 3153–3160.

[41] OpenALPR Technology. 2018. Open Automatic License Plate Recognition Library. http://www.openalpr.com.

[42] OpenCV. 2018. Open Source Computer Vision Library. https://opencv.org.

[43] Silvia Pfeiffer. 2018. *WebVTT: The Web Video Text Tracks Format*. Candidate Recommendation. W3C. https://www.w3.org/TR/2018/CR-webvtt1-20180510/.

[44] Alex Poms, Will Crichton, Pat Hanrahan, and Kayvon Fatahalian. 2018. Scanner: efficient video analysis at scale. *TOG* 37, 4 (2018), 138:1–138:13.

[45] Phongsak Prasithsangaree, Joseph Manojlovich, Jinlin Chen, and Michael Lewis. 2003. UTSAF: a simulation bridge between OneSAF and the Unreal game engine. In *SMC*. 1333–1338.

[46] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: Better, Faster, Stronger. In *CVPR*. 6517–6525.

[47] Fereshteh Sadeghi and Sergey Levine. 2017. CAD2RL: Real Single-Image Flight Without a Single Real Image. In *RSS*.

[48] David Salomon. 2011. *The Computer Graphics Manual*. Springer.

[49] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. 2003. *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550.

[50] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. 2017. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *FSR*. 621–635.

[51] Gary J. Sullivan, Jens-Rainer Ohm, Woojin Han, and Thomas Wiegand. 2012. Overview of the High Efficiency Video Coding (HEVC) Standard. *TCSVT* 22, 12 (2012), 1649–1668.

[52] Xinhui Tian, Shaopeng Dai, Zhihui Du, Wanling Gao, Rui Ren, Yaodong Cheng, Zhifei Zhang, Zhen Jia, Peijian Wang, and Jianfeng Zhan. 2017. BigDataBench-S: An Open-Source Scientific Big Data Benchmark Suite. In *IPDPS*. 1068–1077.

[53] Transaction Processing Performance Council. 2018. TPC-C Benchmark. http://www.tpc.org/tpcc.

[54] Transaction Processing Performance Council. 2018. TPC-E Benchmark. http://www.tpc.org/tpce.

[55] Transaction Processing Performance Council. 2018. TPC-H Benchmark. http://www.tpc.org/tpch.

[56] Longyin Wen, Dawei Du, Zhaowei Cai, Zhen Lei, Ming-Ching Chang, Honggang Qi, Jongwoo Lim, Ming-Hsuan Yang, and Siwei Lyu. 2015. UA-DETRAC: A New Benchmark and Protocol for Multi-Object Detection and Tracking. *arXiv CoRR* abs/1511.04136 (2015).

[57] Thomas Wiegand, Gary J. Sullivan, Gisle Bjntegaard, and Ajay Luthra. 2003. Overview of the H.264/AVC video coding standard. *TCSVT* 13, 7 (2003), 560–576.

[58] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. 2018. BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling. *CoRR* abs/1805.04687 (2018).

[59] Alireza Zare, Alireza Aminlou, Miska M. Hannuksela, and Moncef Gabbouj. 2016. HEVC-compliant Tile-based Streaming of Panoramic Video for Virtual Reality Applications. In *ACMMM*. 601–605.

[60] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodík, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *NSDI*. 377–392.

[61] Shanghang Zhang, Guanhang Wu, João P. Costeira, and José M. F. Moura. 2017. Understanding Traffic Density from Large-Scale Web Camera Data. In *CVPR*. 4264–4273.