Demo: A Query Engine for Zero-streaming Cameras

Mengwei Xu[*]
Peking University
mwx@pku.edu.cn

Tiantu Xu[*]
Purdue ECE
xu944@purdue.edu

Yunxin Liu
Microsoft Research
yunxin.liu@microsoft.com

Xuanzhe Liu
Peking University
xzl@pku.edu.cn

Gang Huang
Peking University
hg@pku.edu.cn

Felix Xiaozhu Lin
Purdue ECE
xzl@purdue.edu

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; • **Information systems** → **Data analytics**.

KEYWORDS

Video Analytics; Zero-streaming Cameras

ACM Reference Format:
Mengwei Xu, Tiantu Xu, Yunxin Liu, Xuanzhe Liu, Gang Huang, and Felix Xiaozhu Lin. 2020. Demo: A Query Engine for Zero-streaming Cameras. In *The 26th Annual International Conference on Mobile Computing and Networking (MobiCom '20), September 21–25, 2020, London, United Kingdom.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3372224.3417317

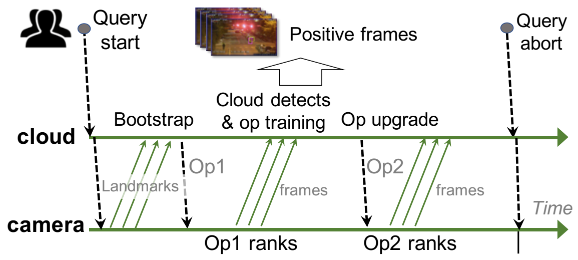

Figure 1: The workflow of a query's execution in ZC^2.

1 INTRODUCTION

Low-cost wireless cameras are growing rapidly. With the help of advanced machine learning models (e.g., CNNs), those videos exhibit high business and social values, e.g., for retailing planning [18], wildlife study [21], and traffic monitoring [19, 25]. However, with high compute need, traditional video analytics systems [14, 15, 26, 27] require all videos to be uploaded to a backend server, which stresses the scarce network bandwidth between cameras and servers.

We advocate for cameras to be **zero-streaming**: a camera captures and stores videos to their cheap local storage without uploading to an edge/cloud server; only when a retrospective query comes, the cloud reaches out to the queried camera. Such advocation is based on two unconventional but practical observations: (1) *Most videos are never queried until expired.* Users typically deploy low-cost cameras for capturing excessive videos [23]. This is because a query typically comes after "interesting" events happen, and those events are often unforeseeable and rare, e.g., traffic accidents or store theft. Without zero streaming, such cold videos waste lots of network bandwidth that are expensive and precious. Provisioning wireless network for cameras to stream such cold videos is uneconomic, and the situation is exacerbated by the increasingly large camera count being deployed. (2) *Camera storage can retain videos long enough.* An increasingly cheap SD card [9, 10] can already store videos (720P at 30fps) for weeks or months. Such retention periods already satisfy many surveillance scenarios [1, 2].

How to query zero-streaming cameras? Since the videos are stored on distributed, remote cameras, existing video analytics systems can no longer suffice. The main challenge is the excessively long query latency: while the cloud compute resource can be easily scaled out (e.g., buying or renting GPUs), the network bandwidth becomes the bottleneck.

We present **ZC^2**, a runtime system that significantly accelerates the query execution for zero-streaming cameras. The key novelty of ZC^2 is to quickly deliver rough query results to users and keep refining the results, a similar concept borrowed from online aggregation [22]. In this way, a user is able to explore the videos through interactive queries, e.g., aborting an ongoing query based on early feedbacks and issuing a new one with parameters updated. Such exploratory query is enabled by several key techniques of ZC^2 (§2).

We build a prototype of ZC^2, which can run on commodity camera hardware (exemplified by Raspberry Pi) and GPU servers. The prototype also contains a web-based GUI, which exhibits rich information to users to examine the query execution progress and results online (§3).

2 ZC^2 DESIGN

Supported queries ZC^2 supports retrospective, ad-hoc video queries that cover a video time span, typically hours or days, and an object class as detectable by modern NNs, e.g., any of the 80 classes of YOLOv3 [24] trained on COCO [17]. Though ZC^2 is designed for rich analytics scenarios including object mean/max/min counting, this demo focuses on a typical one: *retrieving* image frames with certain objects, e.g., "retrieve all images that contain buses from yesterday".

The workflow of ZC^2 contains two following stages.

[*]Equal contributions

During video capture, ZC^2 builds sparse but accurate landmarks with best efforts to obtain long-term video knowledge. The camera runs generic, accurate object detector on a small sample of captured video frames (called *landmarks*). Because the camera hardware (e.g., Arm Cortex-A53) has limited computing capacity, landmarks are sparse, e.g., one in every 30 captured frames; yet, with high-accuracy object labels, they provide reliable knowledge – spatial distributions of various objects over long-term videos. Note that ZC^2 does not use the knowledge as direct answers to queries but builds key query optimizations atop it.

During query execution, ZC^2 runs small NNs (operators) on camera to prioritize frames uploaded and keeps "upgrading" the operators with the help of cloud. As shown in Figure 2, upon receiving a query, the cloud first retrieves all landmarks in the queried video range from camera, along with the object labels and bounding boxes returned by YOLOv3. The cloud uses the landmarks to estimate the object spatial distribution, e.g., "90% of the buses appear in a 200×200 region on the top-right", which is used to optimize the query by running the operators on only this image area. The landmarks are also used as the initial training dataset for bootstrapping a set of camera operators. The camera runs lightweight NNs to prioritize the queried frames for upload. An operator scores frames; a higher score suggests that a frame is more likely to contain any object of interest. The cloud processes the uploaded frames with generic, high-accuracy object detector and emits results, e.g., positive image frames, to users. It trains different operators for higher accuracy. Observing resource conditions (network bandwidth) and positive ratios in uploaded frames, the cloud upgrades the operator on camera. With the upgraded operator, the camera continues to process remaining frames. The above steps repeat until query abort or completion. Throughout the query, the cloud keeps refining the results presented to the user.

3 ZC^2 PROTOTYPE AND DEMO

ZC^2 Prototype We build the cloud runtime atop Tensorflow 1.13 [11] and Keras 2.2.4 [12]. The camera runtime uses Arm NN [3] and generates landmarks with the NNPACK-accelerated YOLOv3 [6]. The cloud uses YOLOv3 to validate the uploaded frames during query execution. Both camera and cloud use OpenCV 3.3 [7] for image processing. We architect on-camera operators as variants of AlexNet [16]. We vary the number of convolutional layers (2–5), convolution kernel sizes (8/16/32), the last dense layer's size (16/32/64); and the input image size (25×25/50×50/100×100). We empirically select 40 operators to be trained by ZC^2 online. As described in §2, the cloud exploits object spatial skews by carving out various image regions for operators to consume. To do so, it employs the k-enclosing algorithm [20] to identify the smallest region that covers a given percentage (e.g., 95%) of the object occurrences. To ensure that on-camera operators will not be bottlenecked by storage or video decoding speed, we use LMDB to accelerate the I/O and also borrow the techniques from VStore [26]. ZC^2 also employs background subtraction [4], a standard technique running on camera during video capture to detect adjacent frames that have little motion (< 1% of the foreground mask) and omits these frames in query execution.

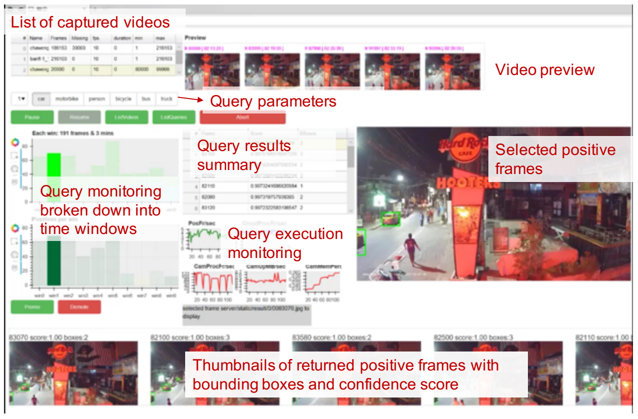
List of captured videos
Video preview
Query parameters
Query results summary
Selected positive frames
Query monitoring broken down into time windows
Query execution monitoring
Thumbnails of returned positive frames with bounding boxes and confidence score

Figure 2: A screenshot of our preliminary demo.

ZC^2 Demo For clarity, this demo only shows the query execution, assuming the landmarks have been built already. The demo runs on Raspberry Pi 3, an embedded hardware similar to low-cost cameras [5, 8], and a commodity x86 server with a modern GPU. Both devices communicates WiFi with 1MB/s default bandwidth [13]. On the cloud/user side, we build a web-based GUI tool (based on Django) to display the information related to query execution.

• **Query initialization in demo** A list of videos will first be displayed with preview frames. The videos are captured ahead of time across different geographical locations, including indoor, traffic, wildlife, etc. The user picks a video, and associated query parameters: time span (e.g., 24 hours), target object class (e.g., bicycle, bus, person), and frame skip (e.g., 1 in 10 frames). Once ready, the query can be issued.

• **Query execution in demo** As query goes on, a set of Web UI fragments will show the query execution progress and results with their UI elements continuously updated. (1) A table that summarizes the returned positive frames. By clicking a table item, the image will be displayed with a bounding box of the target object and a confidence score. (2) A window that displays the query processing monitoring, including positive frames discovered by the system per second, the number of frames being processed by camera/cloud per second, the traffic between camera/cloud, and the memory usage of camera. (3) A window that breaks down the whole queried time range into smaller time windows, and shows the aggregation statistics of each window, such as how many frames have been processed with the current operator. There are buttons to promote/demote certain time windows to adjust the on-camera processing and uploading order if the user favors/disfavors the results returned from that particular window. (4) A window that summarizes the operator is being used for on-camera processing, such as its architecture, speed, and testing accuracy. (5) Buttons to pause or abort the query.

• **Query termination in demo** A query terminates once it's aborted or all frames have been uploaded. All positive frames will be stored on a cloud directory. Besides, the overall query statistics will be summarized such as the query execution time to retrieve 50%/80%/100% positive frames, operators used, etc.

REFERENCES

[1] Tufts: Video security university policy. https://publicsafety.tufts.edu/policies/video-security/, 2014.

[2] Video surveillance laws: Video retention requirements by state. https://www.verkada.com/blog/surveillance-laws-video-retention-requirements/, 2018.

[3] Arm nn ml software. https://github.com/ARM-software/armnn, 2019.

[4] Background subtraction. https://docs.opencv.org/3.4.0/db/d5c/tutorial$_{p}$y$_{b}$g$_{s}$ubtraction.html, 2019.

[5] Hisilicon ip camera specifications. http://www.hisilicon.com/en/Products/ProductList/Surveillance, 2019.

[6] Nnpack-accelerated darknet. https://github.com/digitalbrain79/darknet-nnpack, 2019.

[7] Opencv 3.3. https://opencv.org/opencv-3-3/, 2019.

[8] Wyze camera specifications. https://www.wyze.com/wyze-cam/specs/, 2019.

[9] Price history of 128gb samsung sd card. https://camelcamelcamel.com/product/B06XWZWYVP, 2020.

[10] Price history of 256gb samsung sd card. https://camelcamelcamel.com/product/B072HRDM55, 2020.

[11] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pages 265–283, Savannah, GA, 2016. USENIX Association.

[12] François Chollet. keras. https://github.com/keras-team/keras, 2015.

[13] Bo Han, Feng Qian, Lusheng Ji, and Vijay Gopalakrishnan. Mp-dash: Adaptive video streaming over preference-aware multipath. In Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies, CoNEXT '16, pages 129–143, New York, NY, USA, 2016. ACM.

[14] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B. Gibbons, and Onur Mutlu. Focus: Querying large video datasets with low latency and low cost. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), Carlsbad, CA, 2018. USENIX Association.

[15] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: Optimizing neural network queries over video at scale. Proc. VLDB Endow., 10(11):1586–1597, August 2017.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 25, pages 1097–1105. Curran Associates, Inc., 2012.

[17] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In David J. Fleet, Tomás Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V, volume 8693 of Lecture Notes in Computer Science, pages 740–755. Springer, 2014.

[18] Alan J Lipton, Peter L Venetianer, Niels Haering, Paul C Brewer, Weihong Yin, Zhong Zhang, Li Yu, Yongtong Hu, Gary W Myers, Andrew J Chosak, et al. Video analytics for retail business process monitoring, 2015. US Patent 9,158,975.

[19] Xu Liu, Zilei Wang, Jiashi Feng, and Hongsheng Xi. Highway vehicle counting in compressed domain. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3016–3024, 2016.

[20] Priya Ranjan Sinha Mahapatra, Arindam Karmakar, Sandip Das, and Partha P Goswami. k-enclosing axis-parallel square. In International Conference on Computational Science and Its Applications, pages 84–93. Springer, 2011.

[21] Mohammad Sadegh Norouzzadeh, Anh Nguyen, Margaret Kosmala, Alexandra Swanson, Meredith S Palmer, Craig Packer, and Jeff Clune. Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning. Proceedings of the National Academy of Sciences, 115(25):E5716–E5725, 2018.

[22] Niketan Pansare, Vinayak R Borkar, Chris Jermaine, and Tyson Condie. Online aggregation for large mapreduce jobs. Proc. VLDB Endow, 4(11):1135–1145, 2011.

[23] Ziv Paz. Innovation in surveillance: What's changing at the edge, core and cloud? https://blog.westerndigital.com/innovation-surveillance-edge-core-cloud/, year = 2018.

[24] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767, 2018.

[25] Mengwei Xu, Xiwen Zhang, Yunxin Liu, Gang Huang, Xuanzhe Liu, and Felix Xiaozhu Lin. Approximate query service on autonomous iot cameras. In Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services, pages 191–205, 2020.

[26] Tiantu Xu, Luis Materon Botelho, and Felix Xiaozhu Lin. Vstore: A data store for analytics on large videos. In Proceedings of the Fourteenth EuroSys Conference 2019, EuroSys '19, pages 16:1–16:17, New York, NY, USA, 2019. ACM.

[27] Tan Zhang, Aakanksha Chowdhery, Paramvir (Victor) Bahl, Kyle Jamieson, and Suman Banerjee. The design and implementation of a wireless video surveillance system. In Proceedings of the 21st Annual International Conference on Mobile Computing and Networking, MobiCom '15, pages 426–438, New York, NY, USA, 2015. ACM.