ExSample: Efficient Searches on Video Repositories through Adaptive Sampling

Oscar Moll
MIT CSAIL
Cambridge, MA, USA
orm@csail.mit.edu

Favyen Bastani
MIT CSAIL
Cambridge, MA, USA
favyen@csail.mit.edu

Sam Madden
MIT CSAIL
Cambridge, MA, USA
madden@csail.mit.edu

Mike Stonebraker
MIT CSAIL
Cambridge, MA, USA
stonebraker@csail.mit.edu

Vijay Gadepally
MIT Lincoln Laboratory
Lexington, MA, USA
vijayg@ll.mit.edu

Tim Kraska
MIT CSAIL
Cambridge, MA, USA
kraska@mit.edu

***Abstract*—Capturing and processing video is increasingly common as cameras become cheaper to deploy. At the same time, rich video understanding methods have progressed greatly in the last decade. As a result, many organizations now have massive repositories of video data, with applications in mapping, navigation, autonomous driving, and other areas.**

Because state-of-the-art object detection methods are slow and expensive, our ability to process even simple ad-hoc object search queries ('find 100 traffic lights in dashcam video') over this accumulated data lags far behind our ability to collect it. Processing video at reduced sampling rates is a reasonable default strategy for these types of queries, however, the ideal sampling rate is both data and query dependent. We introduce ExSample, a low cost framework for object search over un-indexed video that quickly processes search queries by adapting the amount and location of sampled frames to the particular data and query being processed.

ExSample prioritizes the processing of frames in a video repository so that processing is focused in portions of video that most likely contain objects of interest. It continually re-prioritizes processing based on feedback from previously processed frames. On large, real-world datasets, ExSample reduces processing time by 1.9x on average and up to 6x over an efficient random sampling baseline. Moreover, we show ExSample finds many most results long before sophisticated, state-of-the-art baselines based on proxy scores can start producing their first results.

***Index Terms*—video data,sampling,object detection**

I. Introduction

Video cameras have become incredibly affordable over the last decade, and are ubiquitously deployed in static and mobile settings, such as smartphones, vehicles, surveillance cameras, and drones. Large video datasets are enabling a new generation of applications. For example, video data from vehicle dashboard-mounted cameras (dashcams) is used to train object detection and tracking models for autonomous driving systems [1]; to annotate map datasets like OpenStreetMap with locations of traffic lights, stop signs, and other infrastructure [2]; and to automate insurance claims processing by analyzing collision scene footage [3].

However, these applications must process large amounts of video to extract useful information. Consider the task of finding examples of traffic lights—to, for example, annotate a map—within a large collection of dashcam video collected from many vehicles. The most basic approach to evaluate this query is to run an object detector frame by frame over the dataset, and select frames where it detects one or more lights. Because state-of-the-art object detectors run at about 10 frames per second (fps) over 1080p video on modern high-end GPUs, scanning through a collection of 1000 hours of 30 fps video with a detector on a GPU would take 3000 GPU-hours. In the offline query case, which is the case we focus on in this paper, we can parallelize our scan over the video across many GPUs, but, as the rental price of a GPU is around $0.50 per hour (for the cheapest $g4$ AWS instance in 2021) [4], our bill for this one ad-hoc query would be $1.5K, regardless of parallelism. Hence, this workload presents challenges in both time and monetary cost. Note that accumulating 1000 hours of video represents just 10 cameras recording for less than a week.

A straightforward means for mitigating this issue is to reduce the number of sampled frames: for example, only run the detector on one frame per second of video. After all, we might think it reasonable to assume all traffic lights are visible for longer than one second. The savings are large compared to inspecting every frame: processing only one frame every second decreases costs by 30x for a video recorded at 30 fps. Unfortunately, this strategy has limitations: for example, the one frame out of every 30 that we look at may not show the light clearly, causing the detector to miss it completely. Second, lights that remain visible in the video for long intervals, like 30 seconds, would be seen multiple times unnecessarily, thereby wasting resources detecting those objects repeatedly. Worse still, we may miss other types of objects that remain visible for shorter intervals, and in general the optimal sampling rate is unknown, and varies across datasets depending on factors such as whether the camera is moving or static, and the angle and distance to the object.

In this paper, we introduce ExSample, a video sampling technique designed to reduce the number of frames that need to be processed by an expensive object detector for object search

queries over large video repositories. ExSample models this problem as one of deciding which frame from the dataset to look at next based on what it has seen in the past. Specifically, ExSample splits the dataset into temporal chunks (e.g, half-hour chunks), and maintains a per-chunk estimate of the probability of finding a new object in a frame randomly sampled from that chunk. ExSample iteratively selects the chunk with the best estimate, samples a frame randomly from the chunk, and processes the frame through the object detector. ExSample updates the per-chunk estimates after each iteration so that the estimates become more accurate as more frames are processed.

Recent state-of-the-art methods for optimizing queries over large video repositories have focused on training cheap, proxy models to approximate the outputs of expensive, deep neural networks [5]–[10]. However, these require training a proxy model for each new query and then running it on the dataset to choose which frames to look at with the more expensive reference detector. While helpful for some queries, for ad-hoc object queries these approaches can impose overheads that exceed the gains. While many of them aim to amortize these costs, the amortization opportunities are constrained to only be for repeated queries involving the same object. Amortization is especially ineffective for limit queries, where the overheads can be so high compared to the query that random sampling can sometimes be used instead.

ExSample addresses the limit query problem from a different perspective, rather than select frames for applying the object detector based on proxy scores, ExSample employs an adaptive sampling technique that eliminates the need for a proxy model and an upfront full scan. A key challenge here is that, in order to generalize across datasets and object types, ExSample must not make assumptions about how long objects remain visible to the camera and how frequently they appear. To address this challenge, ExSample guides future processing using feedback from object detector outputs on previously processed frames. Second, ExSample gives higher weight to portions of video likely to not only contain objects, but also to contain objects that haven't been seen already. Thus, ExSample avoids redundantly processing frames that only contain detections of objects that were previously seen in other frames.

We evaluate ExSample on a variety of search queries spanning different objects, different kinds of video, and different numbers of desired results. We show savings in the number of frames processed ranging up to 6x on real data, with a geometric average of 1.9x across all settings, in comparison to an efficient random sampling baseline. In the worst case, ExSample does not perform worse than `random` sampling, something that is not always true of alternative approaches.

In summary, our contributions are (1) ExSample, an adaptive sampling method that facilitates ad-hoc object searches over video repositories, (2) a formal analysis of ExSample's design, and (3) an empirical evaluation showing ExSample is effective on real datasets and under real system constraints, and that it outperforms existing approaches for object search.

II. Background

In this section we review object detection, introduce distinct object queries, and discuss limitations of prior work.

A. Object Detection

An object detector is a model that operates on still images, inputting an image and outputting a set of boxes within the image containing the objects of interest. The amount of objects found will range from zero to arbitrarily many. Well known examples of object detectors include Yolo [11] and Mask R-CNN [12]. In Figure 1, we show two example frames that have been processed by an object detector, with yellow boxes indicating detections of traffic lights.

Object detectors with state-of-the-art accuracy in benchmarks such as COCO [13] typically execute at around 10 frames per second on modern GPUs, though it is possible to achieve real time rates by sacrificing accuracy [11], [14].

In this paper we do not seek to improve on state-of-the-art object detection approaches. Instead, we treat object detectors as a black box with a costly runtime, and aim to substantially reduce the number of video frames that need to be processed by the detector.

B. Distinct Object Queries

In this paper, we are interested in processing higher level queries on video enabled by the availability of object detectors. In particular, we are concerned with distinct object limit queries such as "find 20 traffic lights in my dataset" over large repositories of multiple videos from multiple cameras. For distinct object queries, each result should be a detection of a different object. For example, in Figure 1, we detected the same traffic light in two frames several seconds apart; although these detections are at different positions and are computed in different frames, in a distinct object query, these two detections yield only one distinct result.

Fig. 1. Two video frames showing the same traffic light instance several seconds apart. A distinct object query is defined by having these two boxes only count as one result.

Then, to specify a query, users must specify not only the object type of interest and the number of desired results, but also a *discriminator function* that determines whether a new detection corresponds to an object that was already seen earlier during processing. In this paper, we assume a fixed discriminator that applies an object tracking algorithm to determine whether a detection is new: it applies a tracker similar to SORT [15] backwards and forwards through video for each detection of a new object to compute the position of that object in each frame where the object was visible;

then, future detections are discarded if they match previously observed positions.

The goal of this paper is to reduce the cost of processing such queries. Moreover, we want to do this on ad-hoc distinct object queries over large video repositories, where there are diverse videos in our dataset and where it is too costly to compute all object detections ahead of time for the type of objects we are looking for. This distinction affects multiple decisions in the paper, including not only the results we return but also the main design of ExSample and how we measure result recall.

Below, we discuss two baselines and prior work for optimizing the processing of distinct object queries.

Naive execution. A straightforward method is to process frames sequentially, applying the object detector on each frame of each video, using the discriminator function to discard repeated detections of the same object. Once we collect enough distinct objects to satisfy the query's limit clause, we can stop scanning. A natural extension is to sample only 1 out of every n frames. Sequential processing exhibits high variance in execution time due to the uneven distribution of objects in video. Moreover, if objects appear in the video for much longer than the sampling rate, we may repeatedly compute detections of the same object. Similarly, if objects appear for shorter than the sampling rate, we may completely miss some objects.

Random sampling. A better strategy is to iteratively process frames uniformly sampled from the video repository (without replacement). This method reduces the query execution time over naive execution as it explores more areas of the data more quickly, whereas sequential execution can get stuck in a long segment of video with no objects. Additionally, early in query execution, randomly sampled frames are less likely to contain previously seen objects compared to frames sampled sequentially.

Proxy-based methods. Methods that optimize video query execution by training cheap, proxy models to approximate the outputs of expensive object detectors have recently attracted much interest. In particular, BlazeIt [10] proposes an adaption of proxy model techniques for processing distinct object queries with limit clauses: rather than randomly sampling frames for processing through the object detector, BlazeIt processes frames in order of the score computed by the proxy model on the frame, beginning with the highest scoring frames. Since the proxy model is trained to produce higher scores on frames containing relevant object detections, this approach effectively ensures that frames processed early during execution contain relevant detections (but not necessarily new objects).

However, proxy-based methods have several critical shortcomings when used for processing object search queries. First, for queries seeking rare objects that appear infrequently in video, these methods require substantial pre-processing to collect annotations for training the proxy models, which can be as costly as solving the search problem in the first place; indeed, BlazeIt resorts to random sampling if the number of positive training labels is too small. Moreover, when processing limit queries, proxy-based methods require an upfront per-query dataset scan in order to compute proxy scores on every video frame in the dataset. As we will show in our evaluation, oftentimes, the cost of performing just this scan is already larger than simply processing a limit query under random sampling.

III. ExSample

In this section we explain how our approach, ExSample, optimizes query execution. Due to the high compute demand of object detectors, runtime in ExSample is roughly proportional to the number of frames processed by the detector. Thus, we focus on minimizing the number of frames processed to find some number of distinct objects.

To do so, ExSample estimates which portions of a video are more likely to yield new results, and samples frames more often from those portions. Importantly, ExSample prioritizes finding distinct objects, rather than purely maximizing the number of computed object detections.

At a high level, ExSample conceptually splits the input into chunks, and scores each chunk separately based on the frequency with which distinct objects have been found in that chunk in the past. This scoring system allows ExSample to allocate resources to more promising chunks while also allowing it to diversify where it looks next over time. In our evaluation, we show that this technique helps ExSample outperform greedy proxy-guided strategies, even when they employ duplicate avoidance heuristics (i.e., do not process frames that are close to previously processed frames).

To make it practical, ExSample is composed of two core components: an estimate of future results per chunk, described in Section III-A, and a mechanism to translate these estimates into a sampling decision which accounts for estimate errors, introduced in Section III-B. In those two sections we focus on quantifying the types of error. Later, in Section III-E, we detail the algorithm step by step.

A. Scoring a Single Chunk

In this section we derive our estimate for the future value of sampling a chunk. To make an optimal decision of which chunk to sample next, assuming n samples have been taken already, ExSample estimates $R(n+1)$ for each chunk, which represents the number of *new* results we expect to find on the next sample. *New* means R does not include objects already found in the previous n samples, even if they appeared also in the $(n+1)^{\text{th}}$ frame. Intuitively, the chunk with the largest $R(n+1)$ is a good location to pick a frame from next.

Our main conceptual tool for reasoning about the quantity $R(n+1)$ as well as our estimates for it throughout this paper is to analyze each result instance in isolation and aggregate to obtain global estimates. More formally, let N be the number of distinct objects in the data. Each object i out of those N is visible for a different number of frames. When sampling frames at random from a chunk, each i will have a different probability p_i of being found proportional to its duration. For

example, in video collected from a vehicle stopped at a red light, red lights will tend to have large p_i lasting in the order of minutes, while green and yellow lights are likely to have much smaller p_i, perhaps in the order of a few seconds. We find in practice these quantities p_i vary widely even for a single object class, from tens to thousands of frames. For any given run of samples, $R(n+1)$ is the sum over the p_i of all as-yet unseen instances $R(n+1) = \Sigma_{i=1}^{N} p_i \cdot [i \notin \text{seen}(n)]$

We emphasize that p_i and N are used in order to reason about ExSample, but they are *not known in advance* by ExSample or by the user. Instead, a key contribution of ExSample is to adapt to a dataset and a query specific set of p_i during the sampling process. Furthermore, we clarify ExSample also does not attempt to estimate individual p_i or N, instead ExSample estimates R directly with the following estimate $\hat{R}$:

$$\hat{R}(n+1) := N^1(n)/n \tag{III.1}$$

Where $N^1(n)$ is the number of distinct *results (objects)* seen exactly once so far, and n is the number of *frames* sampled and inspected so far.

To implement this estimate ExSample tracks how many times we have seen each distinct result. Each result seen only once contributes a count of 1 to $N^1(n)$. Results seen more than once do not contribute anything to $N^1(n)$, and neither do any unseen results (which would be impossible to account for directly). Unlike the p_i or N, which are unknown to the user and to ExSample, $N^1(n)$ is a quantity ExSample can observe.

We now justify the definition of $\hat{R}$ in Eq. III.1 by bounding the expected error $\mathrm{E}[\hat{R} - R]$ in terms of data dependent quantities (we let μ and σ stand for mean and standard deviation)

Theorem (Bias of $\hat{R}$)**.**

$$0 \leq \frac{\mathrm{E}\left[\hat{R} - R\right]}{\hat{R}} \leq \begin{cases} \max p_i \\ \sqrt{N}(\mu_p + \sigma_p) \end{cases} \tag{III.2}$$

Intuitively Eq. III.2 states $\hat{R}$ overestimates R in expectation, that the size of the overestimate is guaranteed to be less than the largest probability, which is likely small, but even if some p_i outliers made $\max p_i$ large, if the standard deviation σ_p is small we can still bound the bias. Eq. III.2 does suggest (but does not imply) that skew (measured by σ_p) could affect the accuracy of the estimate, we will address that experimentally. A large N or a large μ_p may seem problematic for Eq. III.2, but we note that having large number of results N or long average duration μ_p implies many results will be found after only a few samples, so the end goal of searching through the data is easier in the first place and guaranteeing accurate estimates is less important.

Proof. The main proof idea is accounting for each object i's individual expected contribution to both $R(n+1)$ and to $N^1(n)$. We then recover both $N^1(n)$ and $R(n+1)$ by adding each term, justified by linearity of expectation. The individual inequalities of Eq. III.2 follow from analyzing this sum. The event that on a given sequence of samples object i is seen on the $(n+1)^{\text{th}}$ sample after being missed the on the first n samples occurs with probability $p_i(1-p_i)^n$. We denote this quantity $\pi_i(n+1)$. On the other hand, the event that after n samples object i has appeared exactly one occurs with probability $np_i(1-p_i)^{n-1} = n\pi_i(n)$. Therefore $E\left[R(n+1)\right] = \sum \pi(n+1)$, while $E\left[N^1(n)\right] = n\sum \pi(n)$. These sums are taken over the N object indices i, which we will omit for convenience. Therefore $E\left[N^1(n)/n - R(n+1)\right] = \sum \pi(n) - \pi(n+1)$. Because by definition $\pi(n+1) = (1-p)\pi(n)$, the right hand side simplifies to $\sum p\pi_(n)$. Intuitively, term by term this error is small compared to the terms in our estimate: $\sum \pi(n)$, especially when the p_i are small. The expected error is positive, showing the left side of Eq. III.2. The top right inequality is easily seen by replacing the individual p with $\max p$ and factoring it out.

The remaining inequality can be derived by applying Cauchy-Schwartz to the sum.

$$\begin{aligned}
\Sigma p_i \pi_i(n) &\leq \sqrt{(\Sigma p_i^2)(\Sigma \pi_i^2)} && \text{(Cauchy-Schwarz)} \\
&\leq \sqrt{(\Sigma p_i^2)\,(\Sigma \pi_i)^2} && \pi_i \text{ are positive} \\
&= \sqrt{(\Sigma p_i^2)}\Sigma \pi_i \\
&= \sqrt{(\Sigma p_i^2)}\,\mathrm{E}[N^1]/n \\
&= \sqrt{N}\sqrt{(\Sigma p_i^2/N)}\,\mathrm{E}[N^1]/n
\end{aligned}$$

The term $\Sigma p_i^2/N$ is the second moment of the p_i, and can be rewritten as $\sigma_p^2 + \mu_p^2$, where μ_p and σ_p are the mean and standard deviation of the underlying result durations:

$$\begin{aligned}
&= \sqrt{N}\sqrt{(\sigma_p^2 + \mu_p^2)}\,\mathrm{E}[N^1]/n \\
&\leq \sqrt{N}(\sigma_p + \mu_p)\,\mathrm{E}[N^1]/n
\end{aligned}$$

□

B. Picking the best chunk under uncertainty

In this section we explain how we extend the estimate from before with an estimate of its uncertainty so we can use it to make decisions about which chunk to use even when there is some error in our estimates.

If we knew the real R_j for every chunk j, then the optimal algorithm would simply sample from the chunk with the largest R_j value. However, if we use the raw point estimate $\hat{R}_j$ in place of R_j ExSample could get stuck sampling chunks with an early lucky result and ignore better chunks with unlucky early results. [1]

Now we explain how ExSample handles the problem that an observed $N^1(n)$ will fluctuate randomly due to randomness in our sampling. This is especially true early in the sampling process, where only a few samples have been collected but

[1] Beside the uncertainty problem, when sampling from multiple chunks there is also a second, less important concern of how to handle instances spanning multiple chunks. In [16] we show Eq. III.1 can be adjusted to handle this.

we need to make a sampling decision. Because the quality of the estimates themselves is tied to the number of samples we have taken, and we do not want to stop sampling a chunk due to a small amount of bad luck early on, it is important we estimate how noisy our estimate is. The usual way to do this is by estimating the variance of our estimator: $\mathrm{Var}[N^1(n)/n]$, which we find is small compared to $\hat{R}$.

Theorem (Variance)**.**

$$\mathrm{Var}[\hat{R}(n+1)] \leq \mathrm{E}[\hat{R}(n+1)]/n \tag{III.3}$$

Proof. Similar to our estimate of bias, we will estimate the variance $N^1(n)$ assuming independence of the different instances and adding their individual variances. □

We can express $N^1(n)$ a sum of binary indicator variables X_i, which are 1 if instance i has shown up exactly once. $X_i = 1$ with probability $n\pi_i(n)$. Then, because of our independence assumption, the total variance can be estimated by summing. Therefore $N^1(n) = \sum_i X_i$ and because of our independence assumption $\mathrm{Var}[N^1(n)] = \sum_i \mathrm{Var}[X_i]$. Because X_i is a Bernoulli random variable, its variance is $n\pi_i(n)(1-n\pi_i(n))$ which is bounded by $n\pi_i(n)$ itself. Therefore, $\mathrm{Var}[N^1(n)] \leq \sum n\pi(n)$. This latter sum we know is $\mathrm{E}[N^1(n)]$. Therefore $\mathrm{Var}[N^1(n)/n] \leq \mathrm{E}[N^1(n)]/n^2 = \mathrm{E}[\hat{R}(n+1)]/n$.

In fact, we can go further and fully characterize the distribution of values $N^1(n)$ takes, which implicitly tells us about the distribution of $\hat{R}(n)$

Theorem (Sampling distribution of $N^1(n)$)**.** *Assuming p_i are small or n is large, and assuming independent occurrence of instances, $N^1(n)$ follows a Poisson distribution with parameter $\lambda = \Sigma\pi_i(n)$.*

Proof. The proof idea is to treat each instance separately as we did for variance and bias, but this time focus on the moment generating functions. The independence assumption lets us recover the moment generating function of $N^1(n)$ from multiplying the individual functions.

We want to show $N^1(n)$'s moment generating function (MGF) matches that of a Poisson distribution with λ: $M(t) = \exp(\lambda[e^t - 1])$.

As in the proof of Eq. III.3, we think of $N^1(n)$ as a sum of independent binary random variables X_i, one per instance. Each of these variables has a moment generating function $M_{X_i}(t) = 1 + \pi_i(e^t - 1)$. Because $1 + x \approx \exp(x)$ for small x, and $\pi_i(e^t - 1)$ will be small, then $1 + \pi_i(e^t - 1) \approx \exp(\pi_i(e^t - 1))$. Note $\pi_i(e^t - 1)$ is always eventually small for some n because $\pi_i(n+1) = p_i(1-p_i)^n \leq p_i e^{-np_i} \leq 1/en$.

Because the MGF of a sum of independent random variables is the product of the terms' MGFs, we arrive at:
$M_{N^1(n)} = \prod_i M_{X_i}(t) = \exp([\sum_i \pi_i][e^t - 1])$ □

1) Instances spanning multiple chunks: A corner case when deciding which chunk to sample from is how to deal with instances spanning multiple chunks, how do we update the counts?

If instances span multiple chunks, for example a traffic light that spans across the boundaries of two neighboring chunks, Eq. III.1 is still accurate with the caveat that $N_j^1(n_j)$ is interpreted as the number of instances seen exactly once *globally* and which were found in chunk j. The same object found once in two chunks j and k does not contribute to either N_j^1 or to N_k^1, even though each chunk has only seen it once. The derivation of this rule is similar to that in the previous section. In practice, if only a few rare instances span multiple chunks then results are almost the same and this adjustment does not need to be implemented.

At runtime, the numerator N_j^1 will only increase in value the first time we find a new result globally, decrease back as soon as we find it again either in the same chunk or elsewhere, and finding it a third time would not change it anymore. Meanwhile n_j increases upon sampling a frame from that chunk. This natural relative increase and decrease of N_j^1 with respect to each other allows ExSample to seamlessly shift where it allocates samples over time.

Here we prove Eq. III.1 is also valid when different chunks may share instances. Assume we have sampled n_1 frames from chunk 1, n_2 from chunk 2, etc. Assume instance i can appear in multiple chunks: with probability p_{i1} of being seen after sampling chunk 1, p_{i2} of being seen after sampling chunk 2 and so on. We will assume we are working with chunk 1, without loss of generality. The expected number of new instances if we sample once more from chunk 1 is:

$$R_1(n+1) = \sum_{i=1}^{N} \left[p_{i1}(1-p_{i1})^{n_1} \prod_{j=2}^{M} (1-p_{ij})^{n_j} \right]$$

Similarly, the expected number of instances seen exactly once in chunk 1, and in no other chunk up to this point is

$$N_1^1 = \sum_{i=1}^{N} \left[n_1 p_{i1}(1-p_{i1})^{n_1-1} \prod_{j=2}^{M} (1-p_{ij})^{n_j} \right]$$

In both equations, the expression $\prod_{j=2}^{M}(1-p_{ij})^{n_j}$ factors in the need for instance i to not have been while sampling chunks 2 to M. We will abbreviate this factor as q_i. When instances only show up in one chunk, $q_i = 1$, and everything is the same as in Eq. III.1.

The expected error is:

$$N_1^1(n_1)/n_1 - R_1(n_1+1) = \sum_{i=1}^{N} \left[p_{i1}^2 (1-p_{i1})^{n_1-1} q_i \right]$$

Which again is term-by-term smaller than $N_1^1(n_1)/n_1$ by a factor of p_i

C. Thompson Sampling

Now we use the previous results to design a sampling strategy. The goal is to pick among chunks j balancing both the want for a large $\hat{R}_j(n_j)$ and the unreliability of the point estimate when n_j is small. Thompson sampling [17] is one

way to automate this process. Thompson sampling works by modeling unknown quantities such as R_j not just with a point estimate such as $\hat{R}_j$, but with a distribution over its possible values informed by the uncertainty in the estimate. Then, instead of using the point estimate $\hat{R}$ directly to guide decisions, we use a sample from $\hat{R}'s$ distribution. In our implementation, we choose to model the uncertainty of our estimate $\hat{R}_j(n+1)$ as following a Gamma distribution. A Gamma distribution is shaped much like the Normal, except it is restricted to non-negative values. When its mean is a high positive number compared to its standard deviation it appears much like a normal. When its mean is near 0 then its shape changes to have single mode at 0 (we show some different shapes in Figure 2, in solid red). Among other uses, the Gamma distribution is a way to model the uncertainty of the hidden, positive parameter λ of Poisson distribution which we only get to sample. Eq. III-B shows $N^1(n)$ follows a Poisson distribution in many cases, so it is particularly suitable. Rather than mean and standard deviation, the Gamma distribution is typically parametrized by α and β, where both are positive real numbers. We use $\alpha = N_j^1$ and $\beta = n_j$ because the mean is always α/β, or N_j^1/n_j in our case, which is by construction consistent with Eq. III.1, and the variance is always α/β^2, or N_j^1/n_j^2 in our case which is by construction consistent with the bound Eq. III.3. Finally, the Gamma distribution is not defined when α or β are 0, so we need both a way to deal with the scenario where $N^1(n) = 0$ which happens at the start, or when objects are rare, or when only a few objects are left. We do this by adding a small quantity α_0 and β_0 to both terms, obtaining the distribution:

$$R_j(n_j+1) \sim \Gamma(\alpha = N_j^1 + \alpha_0, \beta = n_j + \beta_0) \qquad \text{(III.4)}$$

We used $\alpha_0 = .1$ and $\beta_0 = 1$ in practice, though we did not observe a strong dependence on this value choice. We also experimented with alternatives to Thompson sampling, specifically Bayes-UCB [18], which uses upper quantiles based also on Eq. III.4 instead of samples to make decisions, but we did not observe different results either.

D. Empirical Validation

In this section we provide an empirical validation of Eq. III.1, and Eq. III.4. The question we are interested in is: given an observed N^1 and n, what is the true $R(n+1)$, and how does it compare to the belief distribution $\Gamma(N^1, n)$ from Eq. III.4.

We ran a series of simulation experiments. We first generate 1000 $p_0, p_1, ...p_{1000}$ at random to represent 1000 results with different durations. To ensure there is skew in the p we use a lognormal distribution to generate the p_i. To illustrate the skew in the values, the smallest p_i is 3×10^{-6}, while the $\max p_i = .15$. The parameters μ_p and σ_p are 3×10^{-3} and 8×10^{-3} respectively. For a dataset with 1 million frames (about 10 hours of video), these durations correspond to objects spanning from $1/10$ of a second up to about 1.5 hours, more skew than what normally occurs.

Then we model the sampling of a random frame as follows: each instance is in the frame independently at random with probability p_i. To decide which of the instances will show up in our frame we simulate tossing 1000 coins independently, each with their own p_i, and the positive draws give us the subset of instances visible in that frame. We then proceed drawing these samples sequentially, tracking the number of frames we have sampled n, how many instances we have seen exactly once, N^1, and we also record $\mathrm{E}[R(n+1)]$: the expected number of new instances we can expect in a new frame sampled, which is possible because we can compute it directly as $\Sigma_i^N [i \notin \text{ seen } (n)] \cdot p_i$, because in the simulation we know the remaining unknown instances and know their hidden probabilities p_i, so we compute $\mathrm{E}[R(n+1)]$. We sample frames up to $n = 180000$, and repeat the experiment 10K times, obtaining hundreds of millions of tuples of the form $(n, N^1, R(n+1))$ for our fixed set of p_i. Using this data, we can answer our original question: given an (N^1, n) pair, what is the histogram of the actual $R(n+1)$ across runs and how does it compare to Eq. III.4. We show these histograms for 6 pairs of n and N^1 in Figure 2.

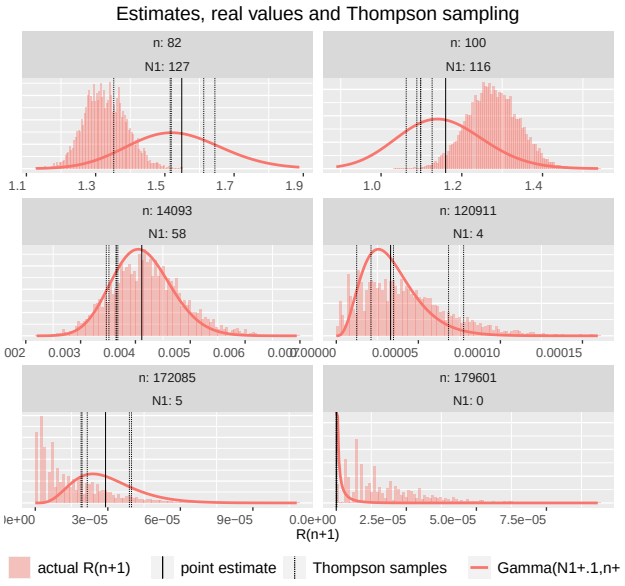

Fig. 2. Comparing our Gamma heuristic of Eq. III.4 with a histogram of the true values $R(n+1)$ from a simulation with heavily skewed p_i. The details of the simulation are discussed in Section III-D. The histograms show the range of values seen for $R(n+1)$ when we have the observed N^1 and n. The point estimate N^1/n of (Eq. III.1) is shown as a solid vertical line. The belief distribution density of Eq. III.4 is plotted as a thicker orange line, and 5 samples drawn from that distribution are shown as dashed vertical lines.

Figure 2 shows a mix of 3 important scenarios. The first 2 subplots with $n \leq 100$, representative of the early stages of sampling. Here we see the Γ model has substantially more variance than the underlying true distribution of $R(n+1)$. This is intuitively expected: when $n = 0$ the value of $R(1)$ is $\sum p$, which our estimator does not know. As n grows to mid range values (next 2 plots), we see that the curve fits the histograms

very well, and also that the curve keeps shifting left to lower and lower orders of magnitude on the x axis. Here we see that the one-sided nature of the Gamma distribution fits the data better than a bell shaped curve. The final 2 subplots show scenarios where n has grown large and N^1 potentially very small, including a case where $N^1 = 0$. In that last subplot, we see the effect of having the extra α_0 in Eq. III.4, which means Thompson sampling will continue producing non-zero values at random and we will eventually correct our estimate when we find a new instance. The bias error is not large enough despite the skew in p.

E. Algorithm

Having derived and demonstrated how to estimate R for different chunks, we now explain ExSample step by step in Algorithm 1, without having any prior information about N or $\mathbf{p}$.

input : video, chunks, detector, discrim, result_limit
output: ans
1 ans $\leftarrow$ [], $N^1 \leftarrow$ [0,0,...,0], $n \leftarrow$ [0,0,...,0]
2 **while** `len(`*ans*`)` $<$ *result_limit* **do**
`// 1) choice of chunk and frame`
3 **for** $j \leftarrow 1$ **to** M **do**
4 $\mathrm{R}_j \leftarrow \Gamma(N^1[j] + \alpha_0, n[j] + \beta_0)$.sample()
5 **end**
6 $j^* \leftarrow \arg\max_j R_j$
7 frame_id $\leftarrow$ chunks[j^*].sample()
`// 2) io,decode,detect,match`
8 rgb_frame $\leftarrow$ video.read_and_decode(frame_id)
9 dets $\leftarrow$ detector(rgb_frame)
`// d0 are the unmatched dets`
`// d1 are dets with only one match`
10 $\mathrm{d}_0, \mathrm{d}_1 \leftarrow$ discrim.get_matches(frame_id, dets)
`// 3) update state`
11 $N^1[j^*] \leftarrow N^1[j^*]$+`len(`$d_0$`)` $-$ `len(`d_1`)`
12 $n[j^*] \leftarrow n[j^*] + 1$
13 discrim.add(frame_id, dets)
14 ans.add(d_0)
15 **end**

Algorithm 1: ExSample

The inputs to the algorithm are:
- `video`: The video data, either a single video or a collection of files.
- `chunks`: How we have partitioned the frames in the video. There are M chunks total.
- `object detector`: Processes video frames and returns object detections (boxes) relevant to the query.
- `discrim`: decides whether a detection is new or matches previous detections.
- `result_limit`: an indication of when to stop.

The loop consists of three parts: picking a frame, processing the frame, and updating the sampler state. To pick a frame ExSample decides which frame to process next. It applies the Thompson sampling step in line 4, where we draw a separate sample R_j from the belief distribution Eq. III.4 for each of the chunks, which is then used in line 6 to pick the highest scoring chunk. As in the rest of the paper, j indexes over chunks. During the first execution of the `while` loop all the belief distributions are identical, but Thompson sampling effectively breaks ties at random. Once decided on a best chunk index j^*, we sample a frame index at random from the chunk in line 7.

Second, ExSample reads and decodes the chosen frame, and applies the object detector on it (line 9). Then, we pass the computed detections on to a discriminator, which compares the detections with objects we have returned earlier during processing in other frames, and decides whether each detection corresponds to a new, distinct object. The discriminator returns two subsets: d_0, the detections that did not match with any previous results (are new objects), and d_1, the detections that matched exactly once with any previous detection. We will only use the size of those sets to update our statistics in the next stage. This frame processing stage of ExSample is the main bottleneck, and is dominated first by the detector call in line 9, and second by the random read and decode in line 8.

Third, we update the state of our algorithm, updating N^1 and n for the chunk we sampled from. We store detections in the discriminator and append the new detections to the result set (`ans`). The amount of tracked state is proportional to the number of chunks, and to the number of results we have detected so far.

F. Other Optimizations

The implementation supplements Algorithm 1 with two optimizations that integrate easily:

Batched sampling Algorithm 1 processes one frame at a time, but on modern GPUs inference throughput is faster when performed on batches of images. The code for a batched version is similar to that in Algorithm 1, but on line 4 we draw B samples per chunk j instead of one sample from each belief distribution. The $\arg\max$ code in Algorithm 1 then produces B batch indices. The distribution over the indices will depend on Thompson sampling. The state update can also be done in batch form: all the updates to N_j^1 and n_j are commutative because they are additive.

Avoiding near duplicates within a single chunk. While `random` sampling is both a reasonable baseline and a good method for sampling frames within selected chunks, `random` allows samples to happen very close to each other in quick succession: for example in a 1000 hour video, random sampling is likely to start sampling frames within the same one hour block after having sampled only about 30 different hours, instead of after having sampled most of the hours once. For this reason, we introduce a variation of random sampling, which we call `random+`, to deliberately avoid sampling temporally near previous samples when possible: by sampling one random frame out of every hour, then sampling one frame out of every not-yet sampled half an hour at random, and so on, until eventually sampling the full dataset. We evaluate the separate

effect of this change in our evaluation. We also use `random+` to sample within a chunk in ExSample, by modifying the internal implementation of the `chunk.sample()` method in line 7.

IV. Determinants of ExSample limits

We have explained how ExSample makes decisions on where to sample and how it adapts its sampling based on statistical estimates. In this section, we explain how much better than random ExSample can be, and when it cannot be much better than random. We first show a different (non practical) sampling method that upper bounds how well ExSample can do. We show though simulation that ExSample matches this upper bound in practice (though only after enough sampling has happened). Second, we identify two important factors that affect how much better than random sampling ExSample can be: the first factor is skew in how results are spread within the dataset, which is inherent to the data, and the second one is the number chunks the data is split into, which the user has picked ahead of time. Using the same simulation we show how these factors affect ExSample performance.

A. Optimal chunk weights

Here we derive an alternative way of sampling chunks that upper bounds ExSample. Though it is not practical, it is helpful to understand when ExSample would not be helpful and also whether ExSample is choosing optimally rather than simply better than random.

ExSample can be thought of as implementing a type of weighted sampling (though the weights are not explicitly computed). At any point during a run, ExSample will have allocated n samples across M different chunks, de-facto assigning a weight of n_j/n to each chunk. A natural question is how this assignment fares against an optimal weight assignment chosen ahead of time (assuming a fixed target n). In this section we derive the optimal weight assignment (as a function of n) which is also a good conceptual benchmark for ExSample. This benchmark is not applicable in real scenarios, but helps understand ExSample and its limits. The number of results discovered by random sampling after n samples follows the curve $N(n) = \sum 1 - (1 - p_i)^n$. If we first split the data into M chunks, conceptually we can think of instance i having an M-dimensional vector $\mathbf{p} = (p_{ij})$, coordinate j in this vector is the conditional probability of seeing instance i when sampling from chunk j. The chance of sampling from a chunk is also an M dimensional vector $\mathbf{w} = (w_j)$, the total chance of sampling instance i is the dot product $\mathbf{p_i w}$. The expected number of instances found after n samples in that scenario is $\sum 1 - (1 - \mathbf{p_i w})^n$. The optimal offline allocation of samples to chunks is therefore

$$\arg\max_{\mathbf{w}} \sum 1 - (1 - \mathbf{p_i w})^n \tag{IV.1}$$

Where $\mathbf{w}$ is constrained to be a valid weight vector (entries are non negatives and add up to 1). In the unrealistic scenario where $\mathbf{p_i}$ for all chunks were known then we can solve for w using a package such as [19]. We later show empirically in Figure 4 that ExSample matches this benchmark as more samples are taken.

Note uniform random sampling corresponds to equal weights and is optimal when all chunks share similar probabilities. However, when some chunks have more results than others or when the p_i change across chunks we expect ExSample will discover better sample allocations.

B. Instance skew across chunks

This section explores how different parameters, in particular skew in instances and average duration of instances, affect ExSample performance in simulation.

Instance skew is a key parameter governing performance of ExSample. If we knew 95% of our results lie on the first half of the dataset, then we could allocate all our samples to that first half. This would boost the p_i of those results in the first half by $2\times$, without requiring knowledge of their precise locations. Hence we could expect about a $2\times$ savings in the frames needed to reach the same number of results. Skew arises in datasets whenever the occurrence of an event correlates relevant latent factor that varies across time e.g., time of day or location (city, country, highway, camera angle) where video is taken. ExSample exploits this skew via sampling when it exists in the data, even though ExSample is not aware of these correlated factors or expects them as inputs.

In our simulation we show that under a wide variety of situations ExSample can outperform `random` by 2x to 80x, and it never does significantly worse. We also explore in detail situations `random` is competitive with ExSample. Results are shown results in Figure 3.

To run the simulation, we fixed the number of instances N to 2000, and the number of frames to 16 million. We placed these 2000 instances according to a normal distribution centered in these 16 million frames, varying the standard deviation to result in no skew (left column in the figure) and skew where 95% of the instances appear in the center 1/4, 1/32, and 1/256 of the frames (additional columns in the figure). To generate varied durations for each of these instances, we use a LogNormal distribution with a target mean of 700 frames. This creates a set of durations where the shortest one is around 50 frames and the longest is around 5000 frames. To change the average duration we simply multiply or divide all durations by powers of 7, to get average durations of $700/49 = 14$, $700/7 = 100$, 700, $700 \times 7 = 4900$. These correspond to the rows in Figure 3.

Once we have fixed these parameters, the different algorithms proceed by sampling frames, and we track the instances that have been found (y axis of subplots) as we increase the number of samples we have processed (x axis). For ExSample, we divide the frames into 128 chunks. We run each experiment 21 times. In Figure 3 we show the median trajectory (solid line) as well as a confidence band computed from the 25th to the 75th percentile (shaded). Additionally, the dashed lines show the expectated N following the optimal allocation for each n, computed using the formula in Eq. IV.1. ExSample

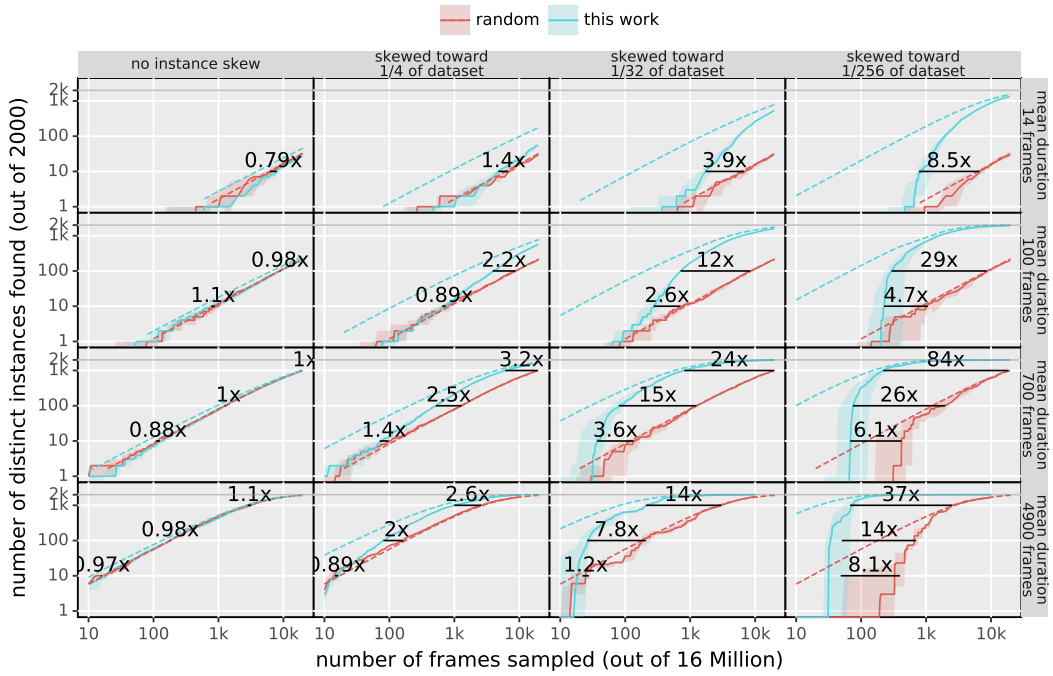

Fig. 3. Simulated savings in frames from ExSample range from 1x to 84x depending depending on instance skew (increasing skew from left to right, starting with no skew), and on the average duration of a instance in frames (increasing from top to bottom). Solid lines show the median frames sampled by ExSample and random. Shaded areas mark 25-75 percentile. We label with text the savings in samples needed to reach 10, 100 and 1000 results. The dashed blue line show the expected results if samples were allocated optimally based on perfect prior knowledge of chunk statistics. The dashed red line shows the expected results from random. For a complete explanation see Section IV-B.

and `random` start off similarly, which is expected because we ExSample starts by sampling uniformly at random, but as samples accumulate, ExSample ś trajectory moves more and more toward the dashed blue line which represents the expected number of instances if the allocation of samples across chunks had been optimal from the beginning. ExSample outperforms random but performs similarly in two cases: 1) when there is very little skew in the locations of results (top row of figure) and or 2) when results are very rare, which makes getting to the first result equally hard for both (left column of figure).

C. Number of chunks

The way we split the dataset into chunks is an external parameter that affects the performance of ExSample. We consider two extremes. Suppose we have a single chunk. This makes ExSample equivalent to random sampling. The fewer chunks there are, the less able to exploit any skew there is. Using two chunks imposes a hard ceiling on savings of $2x$, even if the skew is much larger. On the opposite extreme, for ExSample one chunk per image frame would also be equivalent to random sampling: we would only sample from each chunk once before running out of frames, and we would not be able to use that information to inform later samples.

In Figure 4 we show how ExSample behaves for a range of chunk scenarios spanning several orders of magnitude. In this simulation, we fix instance skew as defined before to 32 and p_i to a mean duration of 700 frames, the same values as the third column and third row of Figure 3. We vary

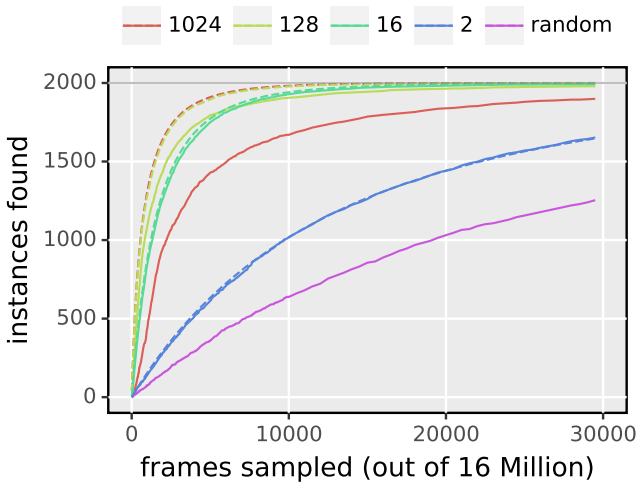

Fig. 4. Varying the number of chunks for a fixed workload. For 2 and 16 chunks, the dashed lines corresponding to optimal sample allocation using Eq. IV.1 match closely with results using ExSample on the simulated data. For 128 and even more so for 1024 chunks there is a noticeable gap between the dashed and solid lines

the number of chunks from 1 to 1024. Dashed lines show the number of instances found when using (static) optimal weights from Eq. IV.1, computed as a function of the x axis. The more chunks there are, the better (higher slope early on) the dashed lines because we have exact knowledge of frame distribution and can exploit skew at smaller time scales. Solid lines in Figure Figure 4 show the median performance of ExSample under the same chunk and data settings, but without

knowledge ahead of time. Notably, increasing the number of chunks can decrease performance: for example going from 128 to 1024 chunks shows a decrease, even though both have similar optimal allocation (dashed) curves. As we increase the number of chunks M, we also increase the number of samples ExSample needs to take just to be able to tell which chunks are more promising: when working with 1024 chunks, just for each chunk to be sampled once we would need to allocate 1024 samples, and this by itself would be too little information about which chunks are more promising. We note that in our simulation we have varied the number of chunks by 3 orders of magnitude and we still see a benefit of chunking versus random across all settings, but we see the benefits are non monotonic.

V. Evaluation

Our goals for this evaluation are to demonstrate the benefits of ExSample on real data compared to alternatives including random sampling and proxy models. We show these challenging datasets ExSample achieves savings of up to 4x with respect to random sampling, and also show that by avoiding the large upfront cost of scoring every frame in the dataset with a query-specific proxy model, ExSample/ is able to find results quicker.

A. Experimental Setup

Baselines. We compare ExSample against two of the baselines discussed in Section II-B: uniform random sampling and a state of the art representative of the proxy model idea (BlazeIt) [10]. BlazeIt is a state of the art video processing system that processes several different types of queries including queries with more complex predicates and aggregates. In this evaluation we are only concerned with how a system like BlazeIt, as a representative of proxy based approaches, handle the specific case of distinct object queries.

Implementation. Random sampling, BlazeIt and ExSample are at their core sampling loops where the choice of which frame to process next is based on an algorithm-specific decision. Random simply picks the next frame randomly. ExSample picks Algorithm 1, and proxy based approaches like BlazeIt pick the next highest scoring frame. The score for every frame is computed by running a proxy model over every frame of the dataset, this proxy model is object specific so the scores cannot be reused for a query for a different object type.

We implement this sampling loop in Python, using PyTorch to run inference on a GPU. For the object detector, we use Faster-RCNN with a ResNet-50 backbone. To achieve fast random access frame decoding rates we use the Hwang library from the Scanner project [20], and re-encode our video data to insert keyframes every 20 frames.

Datasets. We use 5 different datasets in this evaluation, which we call dashcam, bdd, archie, amsterdam and night-street. The dashcam dataset consists of 10 hours of video, or over 1.1 million video frames, collected from a dashcam over several drives in cities and highways. Each drive can range from around 20 minutes to several hours. The BDD dataset used for this evaluation consists of a random sample of 1000 random video clips from the Berkeley Deep Drive Dataset [1]. Becaue BDD video clips are less than 1 minute long, we are forced to use each small clip as an individual chunk. This constraint makes it a challenging scenario for ExSample, as we explained in section IV. The other three datasets: archie, amsterdam and night-steet consist by fixed cameras overlooking urban locations. Each consists of 30 hours of video. These datasets are used in the evaluation of related work on optimizing (other) types of video queries on static camera datasets [6], [10], [21]. There are two important differences between static camera and moving camera settings: in the moving camera setting there is more room for instance skew depending on how the scenery changes over time (for example, the BDD dataset includes data from multiple cities and countries, as well as from highway, suburban and urban settings). For proxy model based approaches, the moving camera setting is also more challenging to create an accurate proxy model for because the background changes constantly and the inputs to the proxy model vary more widely in appearance.

Queries and ground truth We picked between 6 and 8 objects for each of the datasets based on what shows up in them. None of the datasets have human-generated object instance labels that identify objects over time. Therefore we approximate ground truth by sequentially scanning every video in the dataset and running each frame through a reference object detector (we reuse the Faster-RCNN model that we apply during query execution). If any objects are detected, we match the bounding boxes with those from previous frames and resolve which correspond to the same instance. To match objects across neighboring frames, we employ an IoU matching approach similar to SORT [15]. IoU matching is a simple baseline for multi-object tracking that leverages the output of an object detector and matches detection boxes based on overlap across adjacent frames. For all datasets we found we needed to fine-tune the object detector and the iou matching to get to a reasonable quality ground truth. This process involved manually annotated around a hundred frames for each of archie, amsterdam and night-street, retraining Faster-RCNN and then running it over the full datasets to get better quality ground truth. For BDD and dashcam we fine-tuned Faster-RCNN with the BDD labels. For these three datasets we use object types including but not limited to those used in [10].

In addition to searching for different object classes, we also vary the limit parameter recall of 10%, 50% and 90%, where recall is the fraction of distinct instances found. These recall rates are meant to represent different kinds of applications: 10% represents a scenario where an autonomous vehicle data scientist is looking for a few test examples, whereas a higher recall like 90% would be more useful in an urban planning or mapping scenario where finding most objects is desired.

B. Results: sampling immediately vs. proxy scoring overhead

The main advantage of ExSample with respect to using a proxy model is that ExSample requires no prior scoring phase, and therefore avoids a costly scan over the full video dataset. On this first experiment we show the recall level ExSample would achieve in the same amount of time it takes a proxy model to score all frames in the dataset. We show in Figure 5 for most queries ExSample will reach .9 recall before a proxy model has finished scoring the dataset. The remaining queries reach at least .65 in recall. The plot stops at .9 recall because we stop ExSample when it reaches .9 recall.

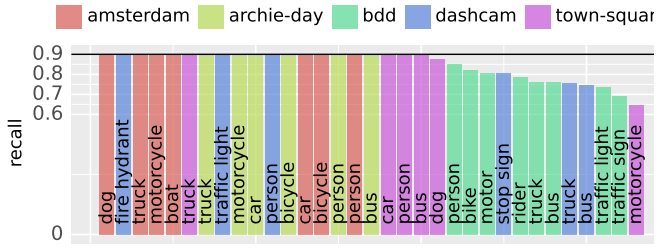

Fig. 5. Recall level for all evaluated queries that ExSample achieves in the same time it takes a proxy model to scan and score the dataset. We stop ExSample when it reaches .9 recall over instances.

The plot in Figure 5 was computed by measuring the scoring throughput we can sustain on our equipment (200 frames per second, bound by io+decode), and then estimating the time it would take to fully scan the dataset. We then estimate how many frames ExSample would be able to sample and process in that time and check how many results it would find, knowing from measurements that ExSample processes frames at a rate of 20 frames per second, bound by the object detector throughput.

Table I shows how long it would take to generate a proxy score on a query for each dataset. Scoring times reach up to 10 hours and this is the reason we prefer to estimate the time rather than implement and run it fully for each of the 34 queries we evaluate as each would take hours to run.

dataset	frames	proxy score time
dashcam	1M	2 h 54 min
bdd1k	330K	55 min
town-square	2.88M	8 h
archie-day	3.54M	9 h 50 min
amsterdam	3.54M	9 h 50 min

TABLE I
PROJECTED PROXY SCORING TIME FOR EACH DATASET

C. Time savings vs. recall level

The previous section showed the time it takes us to compute proxy scores for a whole dataset can be more than enough time for a weighted sampling like ExSample to find most results in the data. Now we are interested in showing how much time ExSample saves on real data to reach different levels of recall compared to `random`, neither of which has a proxy score computation overhead, and hence can produce results immediately. We show results for three levels of recall over instances: .1, .5 and .9 in Figure 6. The maximum is around 6 (leftmost) and the worst case (rightmost) is .75 for boats. The .9 percentile over the 100 bars in the plot is 3.7x and the .1 percentile is 1.2. The geometric mean of savings overall is 1.9 across all vertical bars in the figure.

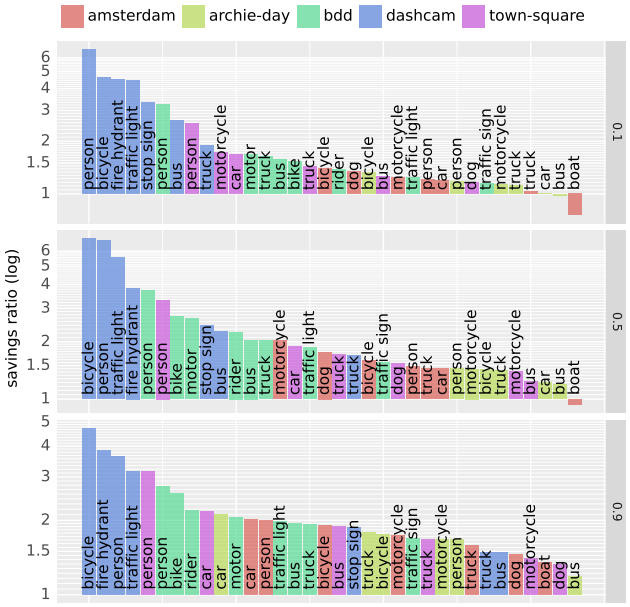

Fig. 6. Time savings when using ExSample vs. `random` for all queries. The top panel corresponds to time savings to reach .10 of all instances.

Figure 7 shows what the chunks for the more extreme cases of Figure 6 look like in terms of skew and abundance. For example, bicycle on the dashcam dataset shows very large skew, which explains why it can achieve results of 3.7x. Then next we have motorcycle in bdd, which even though has very large skew, has relatively low savings of 2x. The reason for the moderate gain is the large amount of chunks in the bdd dataset: it takes a while for ExSample to sample all of them, so it takes a while to notice and take advantage of the skew. We see Figure 6 shows the bar for motorcycle ('motor' under the BDD color) reaches a savings of 3x at recall 5., but has savings of less than 2x early on, likely because it takes time to identify the skew. Finally, 'boat' in the amsterdam dataset, the worse performing query and 'car' in the archie dataset have very low skew, meaning `random`should do just as well. Person in the night street (aka town-square) dataset has moderate amounts of skew and ExSample is able to exploit it.

Figure 6 shows ExSample produces robust gains across 5 datasets and a total of 34 queries. while Figure 5 shows that even a perfect proxy adds too much overhead, so that by the time we have scored frames for a new object class, we could have already sampled a lot of the invidual instances. Part of the reason this is possible is that even the scoring model scores every frame, whereas ExSample samples individual frames,

and avoids sampling frames likely to return existing results. If individual instances last longer, then ExSample will naturally get a large number of instances without needing to look at too many frames. `random`sampling also has this property, but Figure 6 shows ExSample will outcompete random sampling as well.

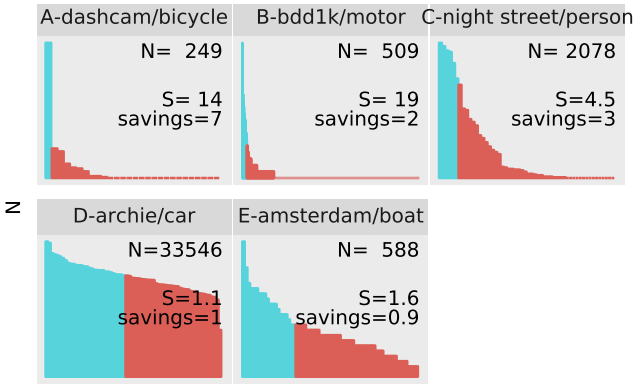

Fig. 7. Instance skew and savings for a few representative queries from 6. Each vertical bar on these plots corresponds to a chunk, and its height is proportional to number of instances. Blue colored bars are the minimum set of chunks that cover half the instances. S is our skew metric defined in Section

VI. Related Work

Video Data Management. There has recently been renewed interest in developing data management systems for video analytics, motivated in large part by the high cost of applying state-of-the-art video understanding methods, which almost always involve expensive deep neural networks, on large datasets. Recent systems adapt prior work in visual data management [22], [23] for modern tasks that involve applying machine learning techniques to extract insights from large video datasets potentially comprising of millions of hours of video. In particular, DeepLens [24] and VisualWorldDB [25] explore a wide range of opportunities that an integrated data management system for video data would enable. DeepLens [24] proposes an architecture split into storage, data flow, and query processing layers, and considers tradeoffs in each layer – for example, the system must choose from several storage formats, including storing each frame as a separate record, storing video in an encoded format, and utilizing a hybrid segmented file. VisualWorldDB proposes storing and exposing video data from several perspectives as a single multidimensional visual object, achieving not only better compression but also faster execution.

Speeding up Video Analytics. Several optimizations have been proposed to speed up execution of various components of video analytics systems to address the cost of mining video data. Many recent approaches train specialized proxy classification models on reduced dimension inputs derived from the video [5]–[9]. As we detailed in Section II-B, most related to our work is BlazeIt [10], which adapts proxy-based video query optimization techniques for object search limit queries.

Video analytics methods employing proxy models are largely orthogonal to our work: we can apply ExSample to sample frames, while still leveraging a cascaded classifier so that expensive models are only applied on frames where fast, proxy models have sufficient confidence. The extensions to limit queries proposed to BlazeIt are not orthogonal, as they involve controlling the sampling method, but a fusion sampling approach that combines ExSample with BlazeIt may be possible. Nevertheless, a major limitation of BlazeIt is that it requires applying the proxy model on every video frame even for limit queries; as we showed in Section V, this presents a major bottleneck that can make BlazeIt slower on limit queries than random sampling.

Besides methods employing proxy models, several approaches, such as Chameleon [21] as well as [26]–[28], consider tuning optimization knobs such as the neural network model architecture, input resolution, and sampling framerate to achieve the optimal speed-accuracy tradeoff. Like proxy models, these approaches are orthogonal to ExSample and can be applied in conjunction.

An alternative approach altogether is to create an index of the dataset ahead of time. For queries on video, this could be done using a multi use proxy embedding model. [29] trains a convolutional model to be reusable for multiple tasks. Part of the motivation is to avoid the repeated per-query scan overhead that makes some of the approaches based on proxy models be very expensive, however it requires some ahead of time knowledge of the queries that will be run. ExSample targets the problem of working with a dataset that has not been indexed ahead of time on ad-hoc queries, usable as long as there is a black box detector to specify what is sought within the dataset.

Speeding up Object Detection. Accelerating video analytics by improving model inference speed has been extensively studied in the computer vision community. Because these methods optimize speed outside of the context of a specific query, they are orthogonal to our approach and can be straightforwardly incorporated. Several general-purpose techniques improve neural network inference speed by pruning unimportant connections [30], [31] or by introducing models that achieve high accuracy with fewer parameters [32], [33]. Coarse-to-fine object detection techniques first detect objects at a lower resolution, and only analyze particular regions of a frame at higher resolutions if there is a large expected accuracy gain [34], [35]. Cross-frame feature propagation techniques accelerate object tracking by applying an expensive detection model only on sparse key frames, and propagating its features to intermediate frames using a lightweight optical flow network [36], [37].

VII. Conclusion

Over the next decade, workloads that process video to extract useful information may become a standard data mining task for analysts in areas such as government, real estate, and autonomous vehicles. Such pipelines present a new systems challenge due to the cost of applying state of the art machine vision techniques over large video repositories.

In this paper we introduced ExSample, an approach for processing distinct object search queries on large video repositories through chunk-based adaptive sampling. Specifically, the aim of the approach is to find frames of video that contain objects of interest, without running an object detection algorithm on every frame, which is often prohibitively expensive. Instead, in ExSample, we adaptively sample frames with the goal of finding the most distinct objects in the shortest amount of time – ExSample iteratively processes batches of frames, tuning the sampling process based on whether new objects were found in the frames sampled on each iteration. To do this tuning, ExSample partitions the data into chunks and dynamically adjusts the frequency with which it samples from different chunks based on the rate at which new objects are sampled from each chunk. We formulate this sampling process as an instance of Thompson sampling, using a Good-Turing estimator to compute the per-chunk likelihood of finding a new object in a random frame. In this way, as objects in a particular chunk are exhausted, ExSample naturally refocuses its sampling on other less frequently sampled chunks.

Our evaluation of ExSample on a real-world dataset of dashcam video shows that it is able to substantially reduce both the number of sampled frames and the execution time needed to achieve a particular recall compared to both random sampling and methods based on lightweight proxy models, such as BlazeIt [10], that are designed to estimate frames likely to contain objects of interest with lower overhead.

REFERENCES

[1] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell, "BDD100K: A diverse driving video database with scalable annotation tooling," *CoRR*, vol. abs/1805.04687, 2018. [Online]. Available: http://arxiv.org/abs/1805.04687

[2] T. Murray, "Help improve imagery in your area with our new camera lending program," https://www.openstreetmap.us/2018/05/camera-lending-program/, 2018.

[3] Nexar, "Nexar," https://www.getnexar.com/company, 2018.

[4] AWS, "Amazon EC2 on-demand pricing," https://aws.amazon.com/ec2/pricing/on-demand, accessed: 2020-09-10.

[5] F. Bastani, S. He, A. Balasingam, K. Gopalakrishnan, M. Alizadeh, H. Balakrishnan, M. Cafarella, T. Kraska, and S. Madden, "Miris: Fast object track queries in video," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1907–1921.

[6] K. Hsieh, G. Ananthanarayanan, P. Bodik, S. Venkataraman, P. Bahl, M. Philipose, P. B. Gibbons, and O. Mutlu, "Focus: Querying large video datasets with low latency and low cost," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 269–286.

[7] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "NoScope: Optimizing neural network queries over video at scale," *Proc. VLDB Endow.*, vol. 10, no. 11, pp. 1586–1597, Aug. 2017. [Online]. Available: https://doi.org/10.14778/3137628.3137664

[8] N. Koudas, R. Li, and I. Xarchakos, "Video monitoring queries," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 1285–1296.

[9] Y. Lu, A. Chowdhery, S. Kandula, and S. Chaudhuri, "Accelerating machine learning inference with probabilistic predicates." ACM SIGMOD, June 2018. [Online]. Available: https://www.microsoft.com/en-us/research/publication/accelerating-machine-learning-queries-with-probabilistic-predicates/

[10] D. Kang, P. Bailis, and M. Zaharia, "Blazeit: Fast exploratory video queries using neural networks," *CoRR*, vol. abs/1805.01046, 2018. [Online]. Available: http://arxiv.org/abs/1805.01046

[11] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[12] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. [Online]. Available: http://arxiv.org/abs/1703.06870

[13] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[14] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," *CoRR*, vol. abs/1611.10012, 2016. [Online]. Available: http://arxiv.org/abs/1611.10012

[15] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," *CoRR*, vol. abs/1602.00763, 2016. [Online]. Available: http://arxiv.org/abs/1602.00763

[16] O. Moll, F. Bastani, S. Madden, M. Stonebraker, V. Gadepally, and T. Kraska, "ExSample: Efficient searches on video repositories through adaptive sampling (technical report)," https://github.com/orm011/tmp/blob/master/techreport.pdf, September 2020.

[17] D. Russo, B. V. Roy, A. Kazerouni, and I. Osband, "A tutorial on thompson sampling," *CoRR*, vol. abs/1707.02038, 2017. [Online]. Available: http://arxiv.org/abs/1707.02038

[18] E. Kaufmann, "On bayesian index policies for sequential resource allocation," *Ann. Stat.*, vol. 46, no. 2, pp. 842–865, Apr. 2018.

[19] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[20] A. Poms, W. Crichton, P. Hanrahan, and K. Fatahalian, "Scanner: Efficient video analysis at scale," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 138:1–138:13, Jul. 2018. [Online]. Available: http://doi.acm.org/10.1145/3197517.3201394

[21] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: Scalable adaptation of video analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 253–266.

[22] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic *et al.*, "Query by image and video content: The QBIC system," *Computer*, vol. 28, no. 9, pp. 23–32, 1995.

[23] V. E. Ogle and M. Stonebraker, "Chabot: Retrieval from a relational database of images," *Computer*, vol. 28, no. 9, pp. 40–48, 1995.

[24] S. Krishnan, A. Dziedzic, and A. J. Elmore, "Deeplens: Towards a visual data management system," in *Conference on Innovative Data Systems Research (CIDR)*, 2019.

[25] B. Haynes, M. Daum, A. Mazumdar, M. Balazinska, A. Cheung, and L. Ceze, "VisualWorldDB: A dbms for the visual world." in *CIDR*, 2020.

[26] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "VideoEdge: Processing camera streams using hierarchical clusters," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 115–131.

[27] R. Xu, J. Koo, R. Kumar, P. Bai, S. Mitra, G. Meghanath, and S. Bagchi, "ApproxNet: Content and contention aware video analytics system for the edge," *arXiv preprint arXiv:1909.02068*, 2019.

[28] T. Xu, L. M. Botelho, and F. X. Lin, "VStore: A data store for analytics on large videos," in *Proceedings of the Fourteenth EuroSys Conference*, 2019, pp. 1–17.

[29] D. Kang, J. Guibas, P. Bailis, T. Hashimoto, and M. Zaharia, "Task-agnostic indexes for deep learning-based queries over unstructured data," Sep. 2020.

[30] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *International Conference on Learning Representations*, 2016.

[31] G. Leclerc, R. C. Fernandez, and S. Madden, "Learning network size while training with ShrinkNets," in *Conference on Systems and Machine Learning*, 2018.

[32] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[33] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer

parameters and 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[34] M. Gao, R. Yu, A. Li, V. I. Morariu, and L. S. Davis, "Dynamic zoom-in network for fast object detection in large images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6926–6935.

[35] M. Najibi, B. Singh, and L. S. Davis, "Autofocus: Efficient multi-scale inference," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 9745–9755.

[36] X. Zhu, J. Dai, L. Yuan, and Y. Wei, "Towards high performance video object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7210–7218.

[37] X. Zhu, Y. Xiong, J. Dai, L. Yuan, and Y. Wei, "Deep feature flow for video recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2349–2358.