ExSample: Efficient Searches on Video Repositories through Adaptive Sampling

Oscar Moll
MIT CSAIL
orm@csail.mit.edu

Favyen Bastani
MIT CSAIL
favyen@csail.mit.edu

Sam Madden
MIT CSAIL
madden@csail.mit.edu

Mike Stonebraker
MIT CSAIL
stonebraker@csail.mit.edu

Vijay Gadepally
MIT Lincoln Laboratory
vijayg@ll.mit.edu

Tim Kraska
MIT CSAIL
kraska@mit.edu

ABSTRACT

Capturing and processing video is increasingly common as cameras improve and 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 example traffic lights in dashboard camera video') over this accumulated data lags far behind our ability to collect it. Processing video at reduced sampling framerates 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 ad-hoc object search over un-indexed video that quickly processes search queries by adapting the amount and location of sampled frames to the data and the 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 up to 4x over an efficient random sampling baseline and by several orders of magnitude over state-of-the-art methods that train specialized surrogate models for each query. ExSample is a key component in building cost-efficient video data management systems.

PVLDB Reference Format:
xxx. xxx. PVLDB, 21(xxx): xxxx-yyyy, 2021.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

1. INTRODUCTION

Proceedings of the VLDB Endowment, Vol. 21, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

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. These 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 [35]; to annotate map datasets like OpenStreetMap with locations of traffic lights, stop signs, and other infrastructure [26]; and to analyze collision scenes in dashcam footage to automate insurance claims processing [28].

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. 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 2020) [1], 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, while the neighboring frames may show it more clearly. 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 will vary 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, surrogate models to approximate the outputs of expensive, deep neural networks [2, 12, 18, 20, 24]. In particular, BlazeIt [17] adapts surrogate-based methods for object search limit queries, and optimizes these queries by training a lightweight surrogate model to output high scores on frames containing the object of interest and low scores on other frames; BlazeIt computes scores through the surrogate over the entire dataset, and then prioritizes high-scoring frames for processing through the object detector. However, this approach has two critical shortcomings, which we expand on in Section 2.2. First, even when processing limit queries, where the user seeks a fixed number of results, BlazeIt still requires a time-consuming upfront dataset scan in order to compute surrogate scores on every video frame. Second, while frames with high surrogate scores are likely to contain object detections relevant to a query, those objects may have already been seen in neighboring frames earlier during processing, and so applying the object detector on those frames does not yield new objects.

ExSample addresses the aforementioned shortcomings to achieve an order of magnitude speedup over the prior state-of-the-art (BlazeIt). First, rather than select frames for applying the object detector based on surrogate scores, ExSample employs an adaptive sampling technique that eliminates the need for a surrogate 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 the outputs of the object detector computed 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.

Sam: xxx update numbers below 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 from 1.1 to 4x, with a geometric average of 2x across all settings, in comparison to an efficient random sampling baseline. Additionally, in comparison to BlazeIt [17], our method processes fewer frames to find the same number of distinct results in many cases, and in the remaining cases ExSample still requires one to two-orders of magnitude less clock time because ExSample does not require an upfront per-query scan of the entire dataset.

Our contributions are (1) ExSample, an adaptive sampling algorithm 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.

2. BACKGROUND

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

2.1 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[31] and Mask R-CNN[10].

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[23] typically execute at around 10 frames per second on modern GPUs, though it is possible to achieve real time rates by sacrificing accuracy [13, 31].

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.

2.2 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 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 have 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 unique result.

Figure 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 [3] 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.

Surrogate-based methods. Methods that optimize video query execution by training cheap, surrogate models to approximate the outputs of expensive object detectors have recently attracted much interest. In particular, BlazeIt [17] proposes an adaption of surrogate 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 surrogate model on the frame, beginning with the highest scoring frames. Since the surrogate model is trained to produce higher scores on frames containing relevant object detections, this approach effectively ensures that frames processed early during execution contain detections.

However, surrogate-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 preprocessing to collect annotations for training the surrogate 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, surrogate-based methods require an upfront per-query dataset scan in order to compute surrogate 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.

3. ExSample

In this section we explain how our approach, ExSample, minimizes the time to finding instances. Due to the high compute demands of object detection models, even when processed in GPUs, runtime and compute cost when using ExSample are approximately proportional to the number of frames processed by the object detector.

In order to minimize the number of frames processed, at a high level, ExSample works by estimating which portions of a video are more likely to yield new results, sampling 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 surrogate-guided strategies, even when they are equipped with duplicate avoidance heuristics (i.e., do not process frames that are temporally 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 3.1, and a mechanism to translate these estimates into a sampling decision which accounts for estimate errors, introduced in Section 3.3. In those two sections we focus on quantifying the types of error. Later, in Section 3.4, we detail the algorithm step by step.

3.1 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, ExSample estimates $R(n+1)$ for each chunk, which represents the number of new results we expect to find on the $(n+1)^{\text{th}}$ sample. New means R does not include objects already found in the previous n samples even if they appear 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.

Sam: Introduce N and p_i independently from the traffic light example.In our traffic light example, the chance of finding a traffic light by random sampling intuitively depends on both the number of traffic lights in the data, which we will call N, and the number of video frames each light i is visible for, which we will call p_i. The chance of finding a new traffic light will depend additionally on how many samples we have drawn already, which we will call n. Note that

p_i varies from light to light: for example in video collected from a moving vehicle that stops 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.

Note that p_i and N are not known in advance. If they were, we could simply define $R(n+1) = \Sigma_i^N [i \notin \text{seen}(n)] \cdot p_i$, where seen$(n)$ is the set of results we have already seen in previous frames. Favyen: Should this not be divided by the number of frames in a chunk?

In the rest of this section we develop and show the correctness of a method for estimating $R(n+1)$. The estimate relies on counting the number of distinct object instances seen exactly once so far, which we denote $N^1(n)$. That is, if a frame is sampled and a new instance is seen once, $N^1(n)$ increases by 1, and then if we saw that instance again, $N^1(n)$ decreases by 1. Unlike the p_i or N, which are unknown to the user and to ExSample, and which will change from dataset to dataset. $N^1(n)$ is a quantity we can observe and track as we sample frames. For a single chunk, our estimate is:

$$R(n+1) \approx N^1(n)/n \tag{1}$$

This formula is similar to the Good-Turing estimator[7], but $N^1(n)$ has a different meaning in those contexts. In our video search application we sample frames, not symbols, at random, and a single frame sample can indirectly contain many objects, or no object at all. This means that in our application, N and $N^1(n)$ could range from 0 to far larger than n itself, for example in a crowded scene. In the traditional use of the Good-Turing estimator (such as the one explained in the original paper [7]) there is exactly one instance per sample and the only question is whether it is new. In that situation $N^1(n)$ will always be smaller than n.

In the remainder of this section we show formally that $N^1(n)/n$ is a good estimate in our setting, and that we can bound its relative bias using high-level properties of the data and the query: the number of distinct result instances N, the average duration of a result $\mu_p = \Sigma p_i/N$ and also the standard deviation of the durations $\sigma_p = \sqrt{\Sigma(p_i - \mu_p)^2/N}$. In this section, error is the bias of our estimate $\text{E}[N^1(n)]/n$, an error that will occur even in the absence of randomness. Sam: Why is E appearing here? Shouldn't the error just be $|R(n+1) - N^1(n)/n|$ In a later section we deal with the problem of errors that arise from randomness in the sampling procedure itself.

In particular we will focus on bounding the relative error:

$$\text{rel. err} = \frac{\text{E}[N^1(n)]/n - \text{E}[R(n+1)]}{\text{E}[N^1(n)]/n}$$

Sam: Again I'm not sure why there is an expectation here?

The following inequalities bound the relative bias error of our estimate:

Theorem (Bias).

$$0 \leq \text{rel. err} \tag{2}$$

$$\text{rel. err} \leq \max p_i \tag{3}$$

$$\text{rel. err} \leq \sqrt{N}(\mu_p + \sigma_p) \tag{4}$$

Intuitively, Equation 2 tells us that $N^1 n/n$ tends to overestimate, Equation 3 that the size of the over-estimate is guaranteed to be less than the largest probability, which is likely small, and Equation 4 says that even if a few p_i outliers were large, as long the durations within one σ of the average μ_p are small the error will remain small. A large N or a large μ_p or σ_p may seem problematic for Equation 4, but we note that a 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 the search problem is easy in the first place and having guarantees of accurate estimates is less important.

In a later experiment with skewed data and many instances we show the estimate works well, and real data in our evaluation has natural skew and we obtain consistently good experimental results.

A proof of Equation 2 Equation 3, and Equation 4 are given in our technical report [?].

3.2 Scoring Multiple Chunks

If we knew $R_j(n_j + 1)$ for every chunk j, the algorithm could simply take the next sample from the chunk with the largest estimate. However, if we simply use the raw estimate:

$$R_j(n_j + 1) \approx N_j^1(n_j)/n_j \tag{5}$$

And pick the chunk with the largest estimate, then we run into two potential problems. The first is the potential problem is the issue of instances spanning multiple chunks. This corner case can be handled easily by adjusting the definition of N_j^1, and can be verified directly in a similar way to Equation 1. The second larger problem is that each chunk's estimate may be off due to the randomness of the sample, especially those chunks with small n_j.

The estimator in this section can be easily adjusted in situations where instances span multiple chunks, and we show the full derivation in the tech report version of this paper.

3.3 Sampling Chunks Under Uncertainty

We assign scores to chunks as we sample them. As we sample, we our goal is to sample each chunk a different number of times, based on the our estimate of the number of new instances in each chunk. We can extend our definition from the previous section to say that n_j in the number of samples of chunk j, each with its own $N_j^1(n_j)$. In this section, we formalize our method for choosing which chunk to sample from next.

Before we develop our sampling method, we need to handle 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 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: $\text{Var}[N^1(n)/n]$. Once we have a good idea on how this variance error depends on the number of samples taken, we can derive our sampling method, which balances both the raw estimate of new objects and the number of samples it is based on.

Theorem (Variance). If instances occur independently of each other, then

$$\text{Var}[N^1(n)/n] \leq \text{E}[N^1(n)]/n^2 \tag{6}$$

Note that this bound also implies that the variance error is more than n times smaller than the value of the raw estimate, because we can rewrite it as $(\text{E}[N^1(n)]/n)/n$. Note

however, that the independence assumption is necessary for proving this bound. While in reality different results may not occur and co-occur truly independently, our experimental results in the evaluation results show our estimate works well in practice.

Proof. We will estimate the variance $N^1(n)$ assuming independence of the different instances, 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 $\pi_i(n) = np_i(1-p_i)^{n-1}$. 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 $\text{Var}[N^1(n)] = \sum_i \text{Var}[X_i]$. Because X_i is a Bernoulli random variable, its variance is $\pi_i(n)(1-\pi_i(n))$ which is bounded by $\pi_i(n)$ itself. Therefore, $\text{Var}[N^1(n)] \leq \sum n\pi_i(n)$. This latter sum we know from before is $\text{E}[N^1(n)]$. Therefore $\text{Var}[N^1(n)/n] \leq \text{E}[N^1(n)]/n^2$. □

In fact, we can go further and fully characterize the distribution of values $N^1(n)$ takes.

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. We prove this by showing $N^1(n)$'s moment generating function (MGF) matches that of a Poisson distribution with λ, $M(t) = \exp\left(\lambda[e^t - 1]\right)$.

As in the proof of Equation 6, 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: $M_{N^1(n)} = \prod_i M_{X_i}(t) = \exp\left(\left[\sum_i \pi_i\right]\left[e^t - 1\right]\right)$. □

3.3.1 Thompson Sampling

Now we can use this bound to design a sampling strategy. The goal is to pick between $(N_j^1(n), n_j)$ and $(N_k^1(n), n_k)$ instead of only between N_j^1 and N_k^1, where the only reasonable answer would be to pick the largest. One way to implement this comparison is to randomize it, which is what Thompson sampling [32] does.

Thompson sampling works by modeling unknown parameters such as R_j not with point estimates such as $N_j^1(n_j)/n_j$ but with a wider distribution over its possible values. The width should depend on our uncertainty. Then, instead of using the point estimate to guide decisions, we use a sample from from this distribution. This effectively adds noise to our point estimate in proportion to how uncertain it is. In our implementation, we choose to model the uncertainty around $R_j(n+1)$ as following a Gamma distribution:

$$R_j(n+1) \sim \Gamma(\alpha = N_j^1, \beta = n_j) \tag{7}$$

A Gamma distribution is shaped much like the Normal when the N^1/n is large, but behaves more like a single-tailed distribution when N^1/n is near 0, which is desirable because N^1/n will become very small over time, but we know our hidden λ is always non-negative. The Gamma distribution is a common way to model the uncertainty around an unobservable parameter λ for a Poisson distribution whose samples we can observe. Because we have shown that $N^1(n)$ does in fact follow a Poisson distribution with an unknown parameter, this choice is especially suitable for our use case. The mean value of the Gamma distribution Equation 7 is $\alpha/\beta = N_j^1/n_j$ which is by construction consistent with Equation 5, and its variance is $\alpha/\beta^2 = N_j^1/n_j^2$ which is by construction consistent with the variance bound of Equation 6.

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 could happen often due to objects being rare, or due to having exhausted all results. As well as at initialization, when both N^1 and n are 0. We do this by adding a small quantity α_0 and β_0 to both terms, obtaining:

$$R_j(n_j+1) \sim \Gamma(\alpha = N_j^1(n_j) + \alpha_0, \beta = n_j + \beta_0) \tag{8}$$

We used $\alpha_0 = .1$ and $\beta_0 = 1$ in practice, though we did not observe a strong dependence on this value. There are alternatives to Thompson sampling such as Bayes-UCB[19], which uses upper quantiles from Equation 8 instead of samples to make decisions, but we did not observe significantly different results.

3.3.2 Empirical Validation

In this section we provide an empirical validation of the estimates from the previous sections, including Equation 5, and Equation 8. The question we are interested in is: given an observed N^1 and n, what is the true expected $R(n+1)$, and how does it compare to the belief distribution $\Gamma(N^1, n)$ which we propose using.

We ran a series of simulation experiments. To do this, we first generate 1000 $p_0, p_1, ...p_{1000}$ at random to represent 1000 results with different durations. To ensure there is duration skew we observe in real data, 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 median p_i is 1×10^{-3}. The parameters μ_p computed from the p_i 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 $\text{E}\left[R(n+1)\right]$: 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 $\text{E}\left[R(n+1)\right]$. We sequentially 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 observed N^1 and n, what is the true expected $R(n+1)$? by selecting different observed n

and N^1 at random, and conditioning (filtering) on them, plotting a histogram of the actual $R(n+1)$ that occurred in all our simulations. We show these histograms for 10 pairs of n and N^1 in Figure 2, alongside our belief distribution.

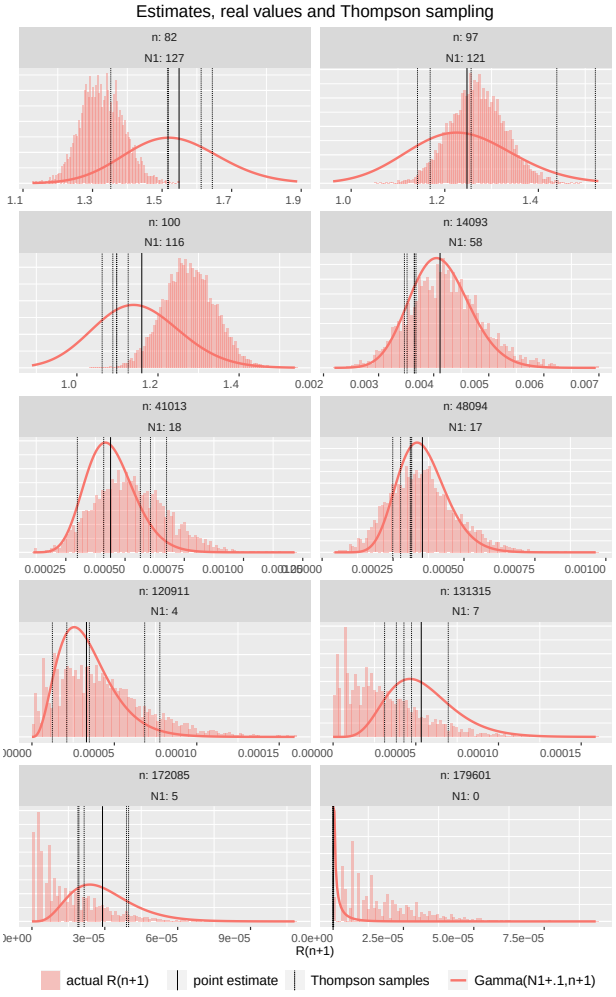


Figure 2: Comparing our Gamma heuristic of Equation 8 with a histogram of the true values $R(n+1)$ from a simulation with heavily skewed p_i. The details of the simulation are discussed in Section 3.3.2. The histograms show the range of values seen for $R(n+1)$ when we have the observed N^1 and n. The point estimate N^1/n of (Equation 5) is shown as a vertical line. The belief distribution density is plotted as a thicker orange line, and 5 samples drawn using Thompson sampling are shown with dashed lines.

Figure 2 shows a mix of 3 important scenarios. The first 3 subplots with $n \leq 100$, representative of the early stages of sampling. Here we see that the Γ model has substantially more variance than the underlying true distribution of $R(n+1)$. This is intuitively expected, because early on both the bias and the variance of our estimate are bottlenecked by the number of samples, and not by the inherent uncertainty of $R(n+1)$. As n grows to mid range values (next 4 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 3 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 Equation 8, 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. In 3 of the subplots there is a clear bias to overestimate, though not that large despite the large skew.

This empirical validation was based on simulated data. In our evaluation we show these modeling choices works well in practice on real datasets as well, where our assumption of independence is not guaranteed and where durations may not necessarily follow the same distribution law.

3.4 Algorithm

The pseudocode for ExSample is shown in Algorithm 1.

```
   input  : video, chunks, detector, discrim, result_limit
   output: ans
1  ans ← []
   // arrays for stats of each chunk
2  N¹ ← [0,0,...,0]
3  n ← [0,0,...,0]
4  while len(ans) < result_limit do
       // 1) choice of chunk and frame
5      for j ← 1 to M do
6          R_j ← Γ(N¹[j] + α0, n[j] + β0).sample()
7      end
8      j* ← arg max_j R_j
9      frame_id ← chunks[j*].sample()
       // 2) io, decode, detection and matching
10     rgb_frame ← video.read_and_decode(frame_id)
11     dets ← detector(rgb_frame)
       // d0 are the unmatched dets
       // d1 are dets with only one match
12     d0, d1 ← discrim.get_matches(frame_id, dets)
       // 3) update state
13     N¹[j*] ← N¹[j*]+len(d0) − len(d1)
14     n[j*] ← n[j*] + 1
15     discrim.add(frame_id, dets)
16     ans.add(d0)
17 end
```

Algorithm 1: ExSample

The inputs to the algorithm are:

- `video`: The video dataset itself, which may be a single video or a collection of files.
- `chunks`: The collection of logical chunks that we have split our video dataset into. There are M chunks total.
- `detector`: An object detector that processes video frames and returns object detections relevant to the query.
- `discrim`: The discriminator function that decides whether a detection is new or redundant.
- `result_limit`: The limit clause indicating when to stop.

After initializing arrays to hold per-chunk statistics, the code can be understood in three parts: choosing a frame, processing the frame, and updating the sampler state. First, ExSample decides which frame to process next. It applies the Thompson sampling step in line 6, where we draw a separate sample R_j from the belief distribution Equation 8 for each of the chunks, which is then used in line 8 to pick

the highest scoring chunk. The j in the code is used as the index variable for any loop over the chunks. During the first execution of the `while` loop all the belief distributions are identical, but their samples will not be, breaking the ties at random. Once we have decided on a best chunk index j^*, we sample a frame index at random from the chunk in line 9.

Second, ExSample reads and decodes the chosen frame, and applies the object detector on it (line 11). 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 11, and second by the random read and decode in line 10.

Third, we update the state of our algorithm, updating N^1 and n for the chunk we sampled from. We also store detections in the discriminator and append the new detections to the result set (`ans`). We note that the amount of state we need to track only grows with the number of results we have output so far, and not with the size of the dataset.

3.5 Other Optimizations

We now extend Algorithm 1 with two optimizations.

Batched execution. Algorithm 1 processes one frame at a time, but on modern GPUs, inference is much faster when performed on batches of images. The code for a batched version is similar to that in Algorithm 1, but on line 6 we draw B samples per j, instead of one sample from each belief distribution, creating B cohorts of samples. In Figure 2 we show 5 different values from Thompson sampling of the same distribution in dashed lines. Because each sample for the same chunk will be different, the chunk with the maximum value will also vary and we will get B chunk indices, biased toward the more promising chunks. The logic for state update only requires small modifications. In principle, we may fear that picking the next B frames at random instead of only 1 frame could lead to suboptimal decision making within that batch, but at least for small values of B up to 50, which is what we use in our evaluation, we saw no significant drop. This is batch size is sufficient to fully utilize a machine with 4 GPUs.

We do not implement or evaluate asynchronous, distributed execution in this paper, but the same reasoning suggests ExSample could be made to scale to an asynchronous setting, with workers processing a batch of frames without waiting for other workers. Ultimately, 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 9.

4. ANALYSIS

We have explained how ExSample allocates samples across chunks. In this section, we develop an analytical model to help us understand the circumstances under which our method performs better than random.

4.1 Upper Bound on Chunking Effectiveness

After n samples, ExSample will have allocated those samples across M different chunks, effectively weighting each chunk differently. In this section we derive an equation parameterized by dataset statistics to derive an upper bound on how well any method that allocates samples across chunks can do in terms of samples taken versus random sampling across the entire data set.

The number of results discovered by random sampling after n samples follows the curve

$$N(n) = \Sigma(1 - (1 - p_i)^n) = N - \Sigma(1 - p_i)^n \tag{9}$$

If we split the dataset into M chunks, then instead of having a single p_i of being found, instance i has a vector $\mathbf{p_i}$ of M entries p_{ij}, where p_{ij} is the conditional probability of instance i being found in a sample on chunk j. Every weighted sampling scheme over these chunks is corresponds to a M dimensional vector $\mathbf{w}$ where w_j is the probability of sampling from chunk j. The total chance of sampling instance i under $\mathbf{w}$ is the dot product $\mathbf{p_i w}$. Hence, $N(n, \mathbf{w}) = N - \Sigma(1 - \mathbf{p_i w})^n$. The optimal allocation of samples to chunks corresponds to the convex problem:

$$N^*(n) = N - \min_{\mathbf{w}} \Sigma(1 - \mathbf{p_i w})^n \tag{10}$$

$$\text{subject to } \mathbf{w} \geq 0 \text{ and } \Sigma w_i = 1. \tag{11}$$

If we assume the M chunks are equally sized, then random sampling corresponds to $\mathbf{w} = \mathbf{1/M}$, and so it is only one of the feasible solutions, which makes clear that optimal weighted sampling is never worse than random sampling, though some implementations of weighted sampling could perform worse than random if weights are chosen poorly.

Note that ExSample does not have access to the $\mathbf{p_i}$ ahead of time, instead ExSample is solving the same optimization problem but in an online fashion. However, for evaluation purposes in this paper, this bound can be computed directly from the $\mathbf{p_i}$ using a solver package such as cvxpy[4], which is what we do in later sections. This bound is useful because once chunks are fixed, no method that relies on weighted random sampling can do better than this curve in expectation, this includes design variations of ExSample using different estimators, or different ways to model uncertainty. We later show empirically that ExSample converges to this bound, depending partially on our choice of chunks.

4.2 Data Dependent Parameters

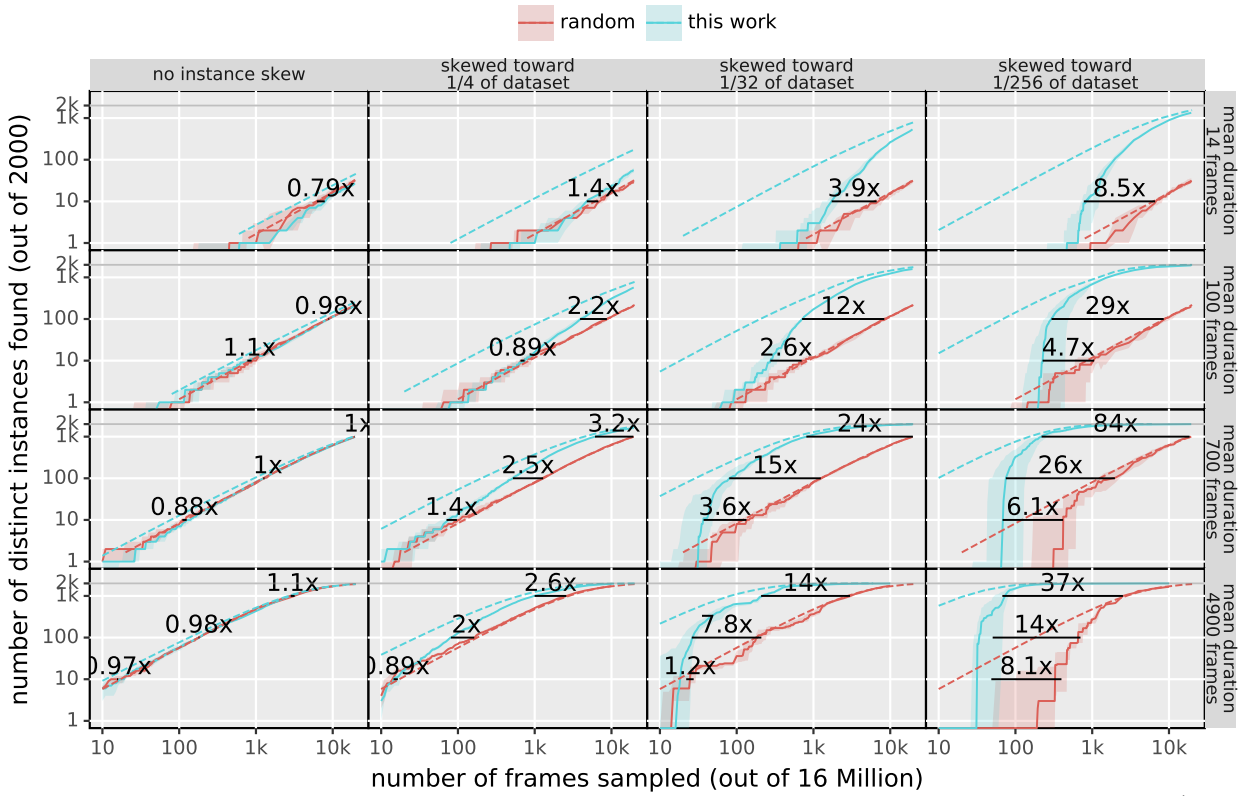


Figure 3: Simulated savings in frames from ExSample range from 1x to 84x depending depending on instance skew (increasing skew from left to right, starting with no skew), and on the average duration of a instance in frames (increasing from top to bottom). Solid lines show the median frames sampled by 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 4.2.

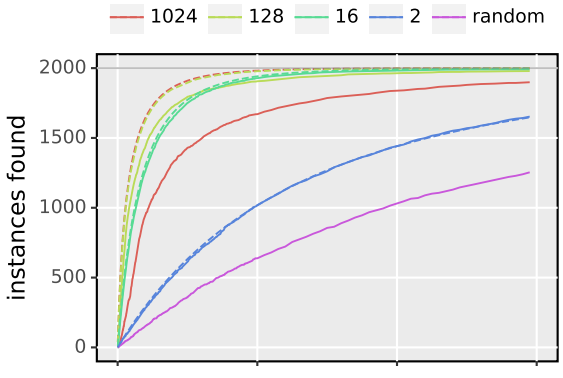
The previous section provided a tight upper bound $N^*(n)$ for any weighted sampling strategy, but it does not guarantee ExSample or even $N^*(n)$ is always better than random sampling. This section explores how different parameters, in particular skew in instances and average duration of instances, 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×, without requiring knowledge of their precise locations. Hence we could expect about a 2× 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. Note that ExSample does not know of these correlated factors or expect them as inputs in any way, but exploits this skew via sampling when it exists in the data.

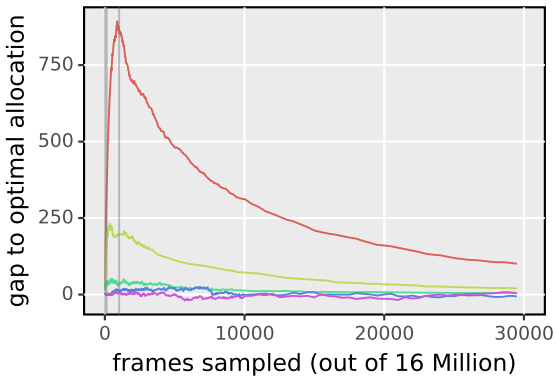
In our simulation we show that under a wide variety of situations ExSample can outperform `random` by 2x to 80x, and it never does significantly worse. We also explore in detail situations `random` is competitive with ExSample. Results are shown results in Figure 3.

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 expectation estimates computed using the formula in ??. ExSample 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



(a) Instances vs samples



(b) Vertical gap between dashed and solid lines

Figure 4: Varying the number of chunks for a fixed workload. Different colors correspond to different number of chunks in the legend. Dashed lines correspond to optimal sample allocation. Solid lines correspond to simulation results.

2) when results are very rare, which makes getting to the first result equally hard for both (left column of figure).

4.3 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, in this case 32. On the opposite extreme, 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 that information to inform the following 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 (32) and p_i (with a mean duration of 700 frames), corresponding to the third column and third row of Figure 3. We vary the number of chunks from 1 to 1024. Dash lines show the optimal allocation from Equation 10; these lines improve with more chunks because we have exact knowledge of frame distribution and can exploit skew at smaller time scales. Note that a linear increase in chunks does not result in linear speedup, because the underlying instance skew is limited.

Solid lines in Figure Figure 4 show the median performance of ExSample under different chunk settings. Here, 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. The reason for this drop is that as we increase the number of chunks, we also increase the number of samples ExSample needs to be able to tell which chunks are more promising: when working with 1024 chunks, we need at least 1024 samples for each chunk to be sampled once. During that time, ExSample performs like random.

The bottom subplot in Figure 4 presents the vertical gap between the dashed line and the solid line in the top subplot. We can see indeed that the gap peaks at around the time the number of samples reaches the number of chunks $n = M$ and decreasing afterwards.

Thus, more chunks M can help improve how much of the instance skew any weighted sampling can exploit, but also implies a larger exploration "tax", proportional to M. For ExSample, this trade off shows up as a longer time to converge to the optimal allocation trajectory with larger M. We note, however that this sensitivity to chunks is not large – in our simulation we vary the number of chunks by 3 orders of magnitude and we still see a benefit of chunking versus random across all settings.

5. EVALUATION

Our goals for this evaluation are to demonstrate the benefits of ExSample on real data, comparing it to alternatives including random sampling as well as existing work in video processing. We show on these challenging datasets ExSample achieves savings of up to 4x Sam: update? with respect to random sampling, and orders of magnitude higher to approaches based on specialized surrogate models.

5.1 Experimental Setup

Baselines. We compare ExSample against two of the baselines discussed in Section 2.2: uniform random sampling and BlazeIt [17]. BlazeIt is the state-of-the-art representative of surrogate model methods for distinct object limit queries, and optimizes these queries by first performing a full scan of the dataset to compute surrogate scores on every video frame, and then processing frames from highest to lowest score through the object detector. As in our approach, BlazeIt uses an object tracker to determine whether each computed detection corresponds to a distinct object.

Implementation. Our implementations of 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 score. Based on this score, the system will fetch a batch of frames from the video and run that batch through an object detector. 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[30], and re-encode our video data to insert keyframes every 20 frames.

We implemented the subset of BlazeIt that optimizes distinct object limit queries, based on the description in the paper [17] as well as their published code. We opted for our own implementation to make sure the I/O and decoding components of both ExSample and BlazeIt were equally optimized, and also because extending BlazeIt to handle our

own datasets and metrics was infeasible. For BlazeIt's cheap surrogate model, we use an ImageNet pre-trained ResNet-18. This model is more expensive than the ones used in that paper, but also more accurate. Note that our timed results do not depend strongly on the inference cost of ResNet-18, since the model is still sufficiently lightweight that BlazeIt's full scan is dominated by video IO and decoding time.

Datasets. We use 5 different datasets in this evaluation. Two datasets consist of vehicle-mounted dashboard camera video, while three datasets are from static street cameras.

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 subset of the Berkeley Deep Drive Dataset[35]. Our subset is made out of 1000 randomly chosen clips. Because the BDD dataset has a wider variety of locations (multiple cities) and is made up of 1000 40-second clips, the number of chunks is forced to be high for this dataset. So, even though both datasets come from dashcams, they test different situations.

The other three datasets are taken from [17], and consist of video recorded by fixed cameras overlooking urban locations. They are named archie, amsterdam and night-street, and each consists of 3 segments of 10 hours each taken on different days. These datasets are used in the evaluations of several existing works on optimizing video queries [12, 16].

Queries. Each dataset contains slightly different types of objects, so we have picked between 6 and 8 objects per dataset based on the content of the dataset. 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.

Ground Truth. 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 [3]. 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.

5.2 Results

Here we evaluate the cost of processing, in both time and frames, distinct object queries using ExSample, `random+random` and BlazeIt. Because some classes are much more common than others, the absolute times and frame counts needed to process a query vary by many orders of magnitude across different object classes. It is easier to get a global view of cost savings by normalizing query processing times of ExSample and BlazeIt against that of `random`. That is, if `random` takes 1 hour to find 20% of traffic lights, and ExSample takes 0.5 hours to do the same, time savings would be 2×. We also apply this normalization when comparing frames processed. In the first part of this section we cover results for dashcam are shown in Figure 5 and for bdd in Figure 6. We cover the remaining datasets later.

5.2.1 Dashcam Dataset Results

Overall, ExSample saves more than 3× in frames processed vs `random`, averaged across all classes and different recall levels. Savings do vary by class, reaching savings of above 5x for the bicycle class in the daschcam dataset. In terms of frames we can see BlazeIt performs really well at recall of 10% in the dashcam dataset, reaching 10× the performance of `random`. The reason is that in the dashcam dataset BlazeIt surrogate models learn to classify frames much better than random, as is evident in Figure 7. For high recall levels, however, this initial advantage diminishes because it is hard to avoid resampling the same instances in different frames, despite skipping a gap of 100 frames when a result is successfully found.

The story becomes more complex when comparing runtime. On the right panel of Figure 5 we show the total runtime costs, including overhead incurred prior to processing the query, which erase the early wins from better prediction on the left panel. The impact of this overhead reduces in magnitude as more samples are taken, and so the difference is less dramatic at higher recall levels. For BlazeIt the total time in the plot is a sum of three separate costs: those of training, scoring, and sampling. Training costs are mostly due to needing to label a fraction of the dataset using the the expensive detector. Scoring costs come from needing to scan and process the full dataset with the surrogate model. Finally, sampling costs come from visiting frames in score order.

In Table 1 we quantify the costs of the three steps, where for BlazeIt we estimate sampling costs by assuming a well trained surrogate model finding instances 100× faster than random. The table shows that BlazeIt's runtime is dominated first by scoring (i.e., the full scan to process every frame through the surrogate model), and second by labeling. While it may be possible to amortize the costs of labeling in BlazeIt across many queries, because the expensive model may provide general-purpose labels, the full scan that dominates the overhead cannot be amortized as the surrogate is query-specific. Because `random` does not need to scan the entire dataset before yielding results, it can afford to be much worse in precision while still providing a faster runtime, especially at low recalls. ExSample outperforms the other methods because it provides high sampling precision while avoiding the need for a full scan.

stage	throughput(fps)	BlazeIt frames/time(s)	random frames/time(s)
label	20	110K/5.8Ks	0
score	100	1.1M/12Ks	0
sample	20	120/5.8s	1.2K/58s

Table 1: Time breakdown: random vs BlazeIt vs ExSample
Sam: add sys

5.2.2 BDD Results

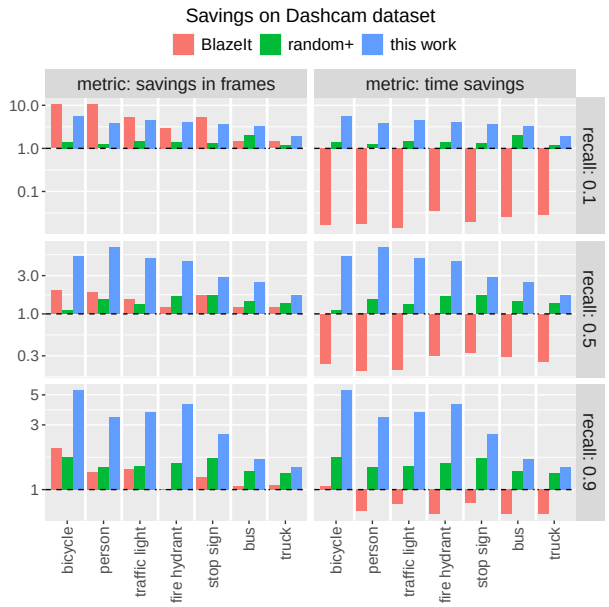


Figure 5: Results on the Dashcam dataset. Comparing frames (left column) and times(right column) for different recall levels (each row) on the dashcam dataset. Different methods (color) are compared relative to the costs of using `random` to process the same query and reach the same level of instance recall.

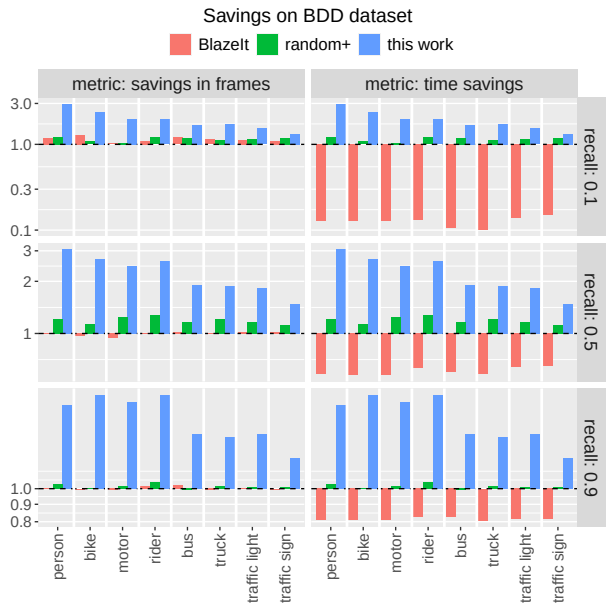


Figure 6: Results on BDD dataset. Surrogate has a harder time learning on this dataset, with lower accuracy scores. Counterintuitively, this makes its savings comparable to those of `random` sampling, improving its results compared to Dashcam datasest

The maximum benefit from ExSample is lower in Figure 6, reaching only 3x. BDD is a challenging scenario for ExSample because as we have mentioned earlier, it is composed of 1000 40 second clips. BDD is also challenging for BlazeIt, because surrogate models have a harder time learning with high accuracy on this dataset, which presents many points of view and small objects. Figure 7 shows that surrogate models do not reach particularly high Average Precision. This is not an issue for ExSample because we do not rely on a cheaper approximation, which may not be as easy to get for highly variable datasets.

5.2.3 *Additional Datasets*

In addition to the Dashcam and BDD datasets, we evaluated ExSample on archie, amsterdam, and night-street from the BlazeIt paper [17], and measured how ExSample performs compared to random in terms of frame savings on all five datasets. This evaluation includes around 35 classes total across the datasets. The results are aggregated in Table 2. Savings are both query and data dependent, and are averaged across all recall levels from 10% to 90%. This is reflected in Figure 8, where we show an object for each dataset, and how its distribution is skewed across chunks in the dataset. The dataset/class combos with the largest skews (dashcam, bdd1k and night-street) also show the largest gains, as expected based on Section 4.2. Amsterdam and archie have little skew overall on any class, hence ExSample cannot extract much more than `random`. In the case of the BDD dataset, the number of chunks is very large and the duration of a clip is short, so the savings are lower than skew would suggest, as explained in Section 4.3.

In addition to comparing across classes, we also compare across recall levels, aggregating across all classes. Overall,

dataset	classes	min (savings)	median	max
dashcam	7	1.7	4.1	6.5
bdd1k	8	1.5	1.9	3.5
night-street	6	1.2	1.6	3.1
amsterdam	7	0.9	1.5	1.7
archie	6	1.2	1.4	1.5

Table 2: Aggregate savings of ExSample relative to random for the different datasets.

the trend is that ExSample performs better at mid-range recall levels. This is because at low recall levels, there have been likely only a few samples and so ExSample has not yet departed as much from `random` allocation.

recall	min (savings)	median	max
10%	0.8	1.4	6.9
50%	0.9	1.7	6.9
90%	1.2	1.9	4.8

Table 3: Aggregate savings broken down by recall.

6. 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 [5, 29] 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 [21] and VisualWorldDB [9] explore a wide range of opportunities that an

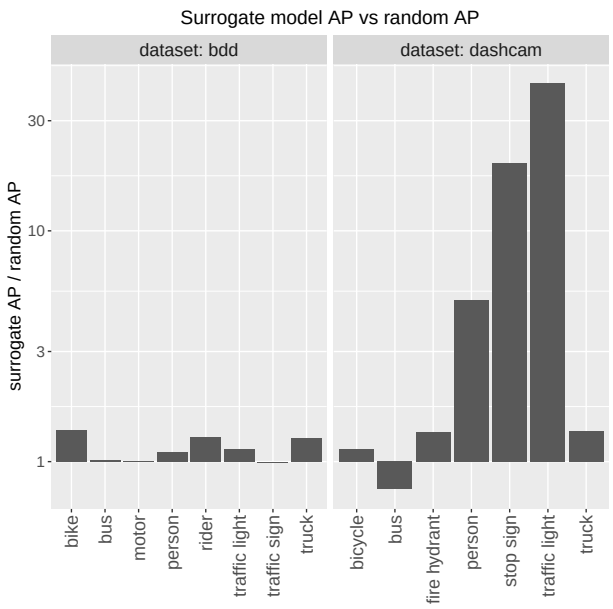


Figure 7: Average Precision of surrogate models vs a randomly assigned score. For BDD, the surrogate model does only slightly better than random in precision, which causes it to perform similarly to random when measured in savings in Figure 6. For the Dashcam dataset, the surrogate models are more accurate than random. Counterintuitively, higher accuracy in scores hurts relative performance in Figure 5 due to the greediness of the algorithm.

integrated data management system for video data would enable. DeepLens [21] 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 surrogate classification models on reduced dimension inputs derived from the video [18, 24, 20, 12, 2]. As we detailed in Section 2.2, most related to our work is BlazeIt [17], which adapts surrogate-based video query optimization techniques for object search limit queries.

Video analytics methods employing surrogate 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, surrogate 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 surrogate model on every video frame even for limit queries; as we showed in Section 5, this presents a major bottleneck that makes BlazeIt even slower on limit queries than random sampling.

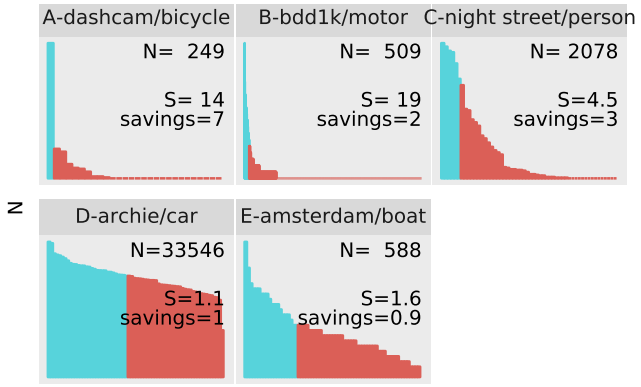


Figure 8: Instance skew and savings for different classes and datasets of Table 2. Each vertical bar corresponds to a chunk's number of instances. The blue colored bars are the minimum set that cover 50% of instances. S is our skew metric defined in Section 4.2.

Besides methods employing surrogate models, several approaches consider tuning optimization knobs such as the neural network model architecture, input resolution, and sampling framerate to achieve the optimal speed-accuracy tradeoff [14, 16, 33, 34]. Like surrogate models, these approaches are orthogonal to ExSample and can be applied in conjunction.

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 [8, 22] or by introducing models that achieve high accuracy with fewer parameters [11, 15]. 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 [6, 27]. 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 [37, 36].

7. CONCLUSION

Over the next decade, workloads that process video to extract useful information may become a standard data mining task for analysts in application 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.

In this paper we introduced ExSample, an approach for processing instance finding queries on large video repositories through chunk-based adaptive sampling. Specifically, the aim of the approach is to find frames of video that contain instances of objects of interest, without running an object detection algorithm on every frame, which could be prohibitively expensive. Instead, in ExSample, we sample frames and run the detector on just the sampled frames, tuning the sampling process based on whether a new instance

of an object of interest is found in the sampled frames. To do this tuning, ExSample partitions the data into chunks and dynamically adjusts the frequency with which it samples from different chunks based on rate at which new instances are sampled from each chunk. We formulate this sampling process as an instance of Thompson sampling, using a Good-Turing estimator to compute the likelihood of finding a new object instance in each video chunk. In this way, as new instances 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 data shows that it is able to substantially improve on both the number of frames it samples and the total runtime versus both random sampling and methods based on lightweight "surrogate" models, such as BlazeIt [17], that are designed to estimate frames likely to contain objects of interest with lower overhead. In particular, these surrogate-based methods are much slower because they require running the surrogate model on all frames.

8. REFERENCES

[1] Amazon ec2 on-demand pricing. `https://aws.amazon.com/ec2/pricing/on-demand`. Accessed: 2020-09-10.

[2] 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, pages 1907–1921, 2020.

[3] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft. Simple online and realtime tracking. CoRR, abs/1602.00763, 2016.

[4] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. Journal of Machine Learning Research, 17(83):1–5, 2016.

[5] 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, 28(9):23–32, 1995.

[6] 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, pages 6926–6935, 2018.

[7] I. J. GOOD. THE POPULATION FREQUENCIES OF SPECIES AND THE ESTIMATION OF POPULATION PARAMETERS. Biometrika, 40(3-4):237–264, 12 1953.

[8] 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.

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

[10] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. CoRR, abs/1703.06870, 2017.

[11] 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, abs/1704.04861, 2017.

[12] 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), pages 269–286, 2018.

[13] 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, abs/1611.10012, 2016.

[14] 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), pages 115–131. IEEE, 2018.

[15] 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.

[16] 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, pages 253–266, 2018.

[17] D. Kang, P. Bailis, and M. Zaharia. Blazeit: Fast exploratory video queries using neural networks. CoRR, abs/1805.01046, 2018.

[18] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia. NoScope: Optimizing neural network queries over video at scale. Proc. VLDB Endow., 10(11):1586–1597, Aug. 2017.

[19] E. Kaufmann. On bayesian index policies for sequential resource allocation. Ann. Stat., 46(2):842–865, Apr. 2018.

[20] N. Koudas, R. Li, and I. Xarchakos. Video monitoring queries. In 2020 IEEE 36th International Conference on Data Engineering (ICDE), pages 1285–1296. IEEE, 2020.

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

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

[23] 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, abs/1405.0312, 2014.

[24] Y. Lu, A. Chowdhery, S. Kandula, and S. Chaudhuri. Accelerating machine learning inference with probabilistic predicates. ACM SIGMOD, June 2018.

[25] D. A. McAllester and R. E. Schapire. On the

convergence rate of good-turing estimators. In Proceedings of the Thirteenth Annual Conference on Computational Learning Theory, COLT '00, pages 1–6, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

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

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

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

[29] V. E. Ogle and M. Stonebraker. Chabot: Retrieval from a relational database of images. Computer, 28(9):40–48, 1995.

[30] A. Poms, W. Crichton, P. Hanrahan, and K. Fatahalian. Scanner: Efficient video analysis at scale. ACM Trans. Graph., 37(4):138:1–138:13, July 2018.

[31] J. Redmon and A. Farhadi. YOLOv3: An incremental improvement. CoRR, abs/1804.02767, 2018.

[32] D. Russo, B. V. Roy, A. Kazerouni, and I. Osband. A tutorial on thompson sampling. CoRR, abs/1707.02038, 2017.

[33] 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.

[34] 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, pages 1–17, 2019.

[35] 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, abs/1805.04687, 2018.

[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, pages 7210–7218, 2018.

[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, pages 2349–2358, 2017.

9. RESPONSE TO REVIEWER COMMENTS

We thank the reviewers for their constructive comments, which have helped us substantially improve our paper.

9.1 Metareview

1. The experimental evaluation needs to be largely strengthened. The authors need to use significantly different datasets that are of much larger size for evaluation, and report additional tests per reviews.. We have added 3 more datasets, used both in BlazeIt and in other papers. These are substantially different in content, they are static (vs moving camera in our case), they are larger in size (30 hours each), and they are made up of a single camera stream, rather than independent clips. We test a wide range of queries on them, bringing the total number of queries to 35. Additionally, we include extension simulation experiments in section 4 designed to highlight properties of these datasets that allow ExSample to work well.
2. Substantial improvements on writing are needed. We apologize, and have revised our writing to make it easier to read. Sam: Can we say anything concrete about what we have done.
3. Additional discussions on related work are needed. We expanded our discussion of related work to include suggestions from the reviewers, including more adjacent work from both the systems community and the computer vision community in a greatly expanded related work section.
4. More discussions and in-depth analysis of the proposed approach are needed, including when the system may fail and other issues pointed out in the reviews. In addition to expanding our evaluation, we greatly expanded our discussion of ExSample upsides, trade-offs and limitations with simulations over synthetic datasets as well as mathematical arguments in a new section (section 4).

9.2 Reviewer 1

9.2.1 Weakpoints and Detailed Evaluation

1. The main shortcoming to me is the evaluation. It's done on very limited datasets, two of the exact same kind. Little can be inferred from this. They are also very small.. We have added more datasets in subsubsection 5.2.3. We also explain why the existing datasets are more different than they may appear, because of the effect of having very many small video files in BDD. The effect of having many files is explored through simulation in a new section subsection 4.3. We have also added a new section on simulation based-analysis, which is in subsection 4.2, which helps to shed light on when ExSample performs well versus random sampling and when it does not.
2. Further, the presentation lacks a critical analysis of the approach. Where does the approach, i.e., under what circumstances, does the approach fail? Using entirely different datasets (or somehow synthetic datasets) could shed a light on this. We have added experiments on synthetic datasets to better characterize this. See item 4 in subsection 9.1.
3. Another issue which I did not understand well, is how the p's are defined for different objects? Does the user ultimately need to provide them? How are they determined? How does speed influence p? Again, using pretty much the same kind of video for this is not helpful at all. What p's were used for the experiments? Will they be significantly different for other objects? The user does not need to provide p, and ExSample does not assume prior knowledge of these values. We wholly agree with your opinion that the usefulness of ExSample would be much reduced otherwise: type of object, speed of object or camera motion speed would invalidate assumptions. We have updated our writing to make this clear early on, and in subsection 4.2) we vary the p_i several orders of magnitude, and show ExSample works. This is one of the strengths of ExSample.
4. Overall, I found the method quite simple. It's rather intuitive and, in terms of cleverness, marginally better than random sampling. We agree that the ExSample algorithm is simple; that is part of its appeal and it is exciting to us that such a simple method can frequently outperform competing methods published in VLDB/SIGMOD. We believe our core contributions are 1) providing a strong mathematical foundation that shows why and when ExSample will help, and why ExSample will not make things worse (which cannot be said about competing methods, as our results show!) and 2) not requiring heavy prior knowledge about the data (either in the form of user configuration, hard-coded assumptions in the model, or pre-trained specialized models), nor strong assumptions about the nature of data such as whether the camera is fixed or it moves, or if events last a fixed amount of time, and without requiring a scan of the full dataset prior to starting.

9.3 Reviewer 2

9.3.1 Weak points

1. For distinct object query, the algorithm presented follows the routine of "sample frame -> detect object -> matching check". However, it is not clear whether there are some existing efforts in the computer vision field that are tailored for distinct object detection or dynamic object detection in video, instead of the static object detection used in the paper? For example, would motion detection or tracking technique more suitable for distinct object query so that we can avoid matching check? We have added some discussion of object detection in video as part of our related work section. Existing efforts in the CV community for dynamic object detection in video emphasize improving accuracy of detection by leveraging use of multiple frames at the same time, or some kind of temporal context, for each prediction. These are not necessarily cheaper and they would be orthogonal to what we do. Stand-alone tracking is cheaper than detection, but requires a scan of the frames. In Table 1 we show scanning with a light weight per-frame algorithm is not necessarily cheap. Hence, the rationale behind avoiding tracking is similar to the rationale behind avoiding scanning.
2. It is not clear under which circumstances ExSample outperforms other algorithms.
   (a) For instance, with large recall (in LIMIT), would sequential execution be better than ExSample, especially with dynamic object detection or motion detection in the video?. The simulation section (subsection 4.2) shows more clearly ExSample helps save frames at every level of recall, and may struggle most

early on. This is also shown in real data in Table 3. For extremely high recall, if we have no constraints on minimum duration of an event, then all methods would need to process most of all the frames. Whether this is done sequentially is no longer important, because the dominant cost is the expensive classifier.

(b) When there are objects that only appear for a very short period, any random sampling scheme is going to be hard to capture such objects. If the query asks for large recall (in LIMIT), sequential execution is preferred in this adversary case.
Our simulation explores this case of objects with very short duration, and we better characterize the cases when ExSample will and will not outperform random (or sequential).

3. How much is the novelty of Bias and variance calculation? What is the bias and variance for the original Good-turing estimator? Is it similar to the one in the paper? The main source we looked at for the original GT estimator is [25]. The main result in [7] as it relates to our use case is about the estimator having low bias. Neither of these papers cover the situation we face when sampling frames. Our contribution is of estimator to work with the use cases of ExSample we extend it in 3 ways 1.) we verify we can adapt it to a scenario where multiple results can appear in one sample 2.) We related the bias to statistics of the query and data (μ_p and σ_p) 3.) we model the uncertainty as n grows so we can use it in a bandit context 4.) we verify we can cover case where a single instance may appear in different chunks. We have expanded our presentation of this analysis to clarify this.
4. In the experiments, I think it may not be fair to include the pre-processing time of BlazeIt – the pre-processing time can be amortized across different queries (not only limit-k distinct object query) and can even be transferred to a different dataset. We presented two views of the results. One uses frames processed as a metric. This metric excludes all preprocessing, and is a proxy for time (all of ExSample and BlazeItand `random`will essentially sample frames from the dataset and run an expensive classifier. This is the view in the left panels of Figure 6 and Figure 5. On the right panels we include pre-processing costs. The preprocessing cost is due to both training and scanning the whole dataset. In the plots we showed, more than 50% of the overhead is due to scanning the dataset. We added Table 1 to make this clear. The scanning section is done per-query, i.e., it cannot be amortized across queries. For any new type of query we need to score every frame after training the specialized model. The training section can be amortized if any data that arrives is assumed to be similar to old data. This could be true of static cameras. Datasets with multiple moving cameras change more, and so it is more of an open question how much one can amortize even the training while keeping the lightweight model lightweight. The results on the BDD dataset suggest a lightweight model is not always a given. We think all of these different scenarios are important depending on the use case, and no one metric can capture these trade offs.
5. Why not using the same dataset as that in BlazeIt? We have now added these in subsubsection 5.2.3. They were not publicly available as of the first version of this paper.
6. Also, more baselines – maybe the other approaches like Noscope and PP[12] have better performance than BlazeIt for distinct object queries? We originally were comparing against Noscope, but found BlazeIt to be a better approach. The main issue is that Noscope is inherently sequential, so it is much more likely than BlazeIt to keep applying the expensive detector on the same instance. BlazeIt also has an advantage can decide where to sample next with a global view of where the highest scores are. Noscope has to decide ahead of time on a threshold and may miss instances entirely due to this. When we tried adapting Noscope to our use case it started to look more and more like BlazeIt, and consistently performed worse.

9.3.2 Detailed evaluation

1. For Thompson sampling, a natural alternative is to formulate it as a multi-arm bandit problem and use upper confidence bound (UCB) as the strategy. Have you considered this alternative and how would this perform compared to Thompson sampling? Thank you for the recommendation. We evaluated this alternative, specifically Bayes-UCB[19], using the same Gamma distribution model derived in subsection 3.3, and we observed almost identical trajectories with those of TS in the simulation in subsection 4.2. Part of the reason may be that subsection 4.1 shows an exact upper bound for any weighting scheme based on chunks, which thus applies to both Thompson sampling and Bayes-UCB variants, as well as any other alternatives, and in subsection 4.2 we observe ExSample converges to the optimal allocation fairly quickly in many cases, so that the gains from any alternative bandit strategy are bound to be small especially at higher recall levels.
2. Is the video stored on Disk? how is the time spent on random access to different frames? We find that the expensive object detector (20 fps) dominates random access as a cost. We measured this on a 1 GPU 1 SSD setup, and requires keyframes to be inserted every 10 frames. Being able to sample frames in batches and to update chunk statistics asynchronously for different batches means ExSample can in principle exploit parallel I/O, as well as overlap parallel I/O with parallel compute.
3. It would be good to have S3.3.2 not only on the synthetic dataset but also on the real dataset in S4. To help bridge the gap between simulation and real data, we added section 4 and Figure 3 as well as Figure 4, and these simulated results mostly help explain the bottom line in the evaluation. We hope those changes address the underlying goal of this request.
4. Lots of typos. We apologize for that and thank you for your patience when reading through our typos in the previous manuscript. We have proof-read more carefully this time.

9.3.3 Revision

1. Discussion on any existing efforts in the computer vision field that are tailored for distinct object detection or dynamic object detection in video, instead of the static object detection used in the paper. We have expanded our related work section to discuss this.section 6
2. Discussion on when ExSample may fail. We added section 4 to address this, and also some new datasets in

section 5.

3. In the experiment, present the time of BlazeIt without pre-training time; and use datasets in BlazeIt. We have added 3 of 6 datasets in BlazeIt. 2 of the remaining essentially have only one class of query (boat) and in those datasets there are boats everywhere all the time, because BlazeIt targets counting queries as a workload, or queries such as 'return frames where there are more than 7 boats'. The plots in Figure 5 and Figure 6 present on the left frames, which are a proxy for time without any pre-training or scanning, and in Table 1 we break down pre-training from scanning.

9.4 Reviewer 3

9.4.1 Detailed Evaluation and Revision

1. The paper is not very well written, neither well explained. It's difficult to follow the idea of the authors. We apologize for the lack of clarity in some sections. We have used your comments to improve the writing, and believe you have helped us make the exposition clearer this time around.
2. There are some format errors. Please proofread the manuscript carefully. For example, (i) In page 1, the citation item in the sentence '… as the rental price of a GPU is around $3 per hour[?] is broken. (ii) In page 12, the content exceeds the page margin. (iii) Fig. 1 is blank. We apologize for these and thank you for your patience with these issues while reviewing our manuscript. We have been more careful with proofreading this new version.
3. This paper studies the problem of querying on large video. However, some important related works (e.g., [1,2]) in this area are missing. [1] K. Hsieh et al., Focus: Querying large video datasets with low latency and low cost. OSDI18. [2] J. Jiang et al., Chameleon: scalable adaptation of video analytics. SIGCOMM18 We have added a discussion of these two papers in our related work section, as well as other work. Thank you for bringing them to our attention.