BlazeIt: Fast Exploratory Video Queries using Neural Networks

Daniel Kang, Peter Bailis, Matei Zaharia
Stanford InfoLab

ABSTRACT

As video volumes grow, analysts have increasingly turned to deep learning to process visual data. While these deep networks deliver impressive levels of accuracy, they execute as much as 10× slower than real time (3 fps) on a state-of-the-art GPU, which is infeasible at scale (8 decades of GPU time for 100-camera months). In addition, deploying these methods requires writing complex, imperative code with many low-level libraries (e.g., OpenCV, MXNet), an often ad-hoc and time-consuming process. To address the computational and usability challenges of video analytics at scale, we introduce BLAZEIT, a system that optimizes queries over video for spatiotemporal information of objects. BLAZEIT accepts queries via FRAMEQL, a declarative language for exploratory video analytics, that enables video-specific query optimization. We propose new query optimization techniques uniquely suited to video analytics that are not supported by prior work. First, we adapt control variates to video analytics and provide advances in specialization for aggregation queries. Second, we adapt importance-sampling using specialized NNs for cardinality-limited video search (i.e. scrubbing queries). Third, we show how to infer new classes of filters for content-based selection. By combining these optimizations, BLAZEIT can deliver over three order of magnitude speedups over the recent literature on video processing.

PVLDB Reference Format:
Daniel Kang, Peter Bailis, Matei Zaharia. BlazeIt: Fast Exploratory Video Queries using Neural Networks. *PVLDB*, 11 (5): xxxx-yyyy, 2018.
DOI: https://doi.org/TBD

1 Introduction

Video is rich with semantic information and is a rapidly expanding source of data at scale. For example, London alone has over 500,000 CCTVs [2], and a single autonomous vehicle can generate terabytes of data per day [23]. This growing volume of video can provide answers to queries about the real world. Thus, analysts are increasingly interested in running *exploratory queries* to quickly understand higher-level information over video. For example, an urban planner working on traffic meter setting [71] or urban planning [9] may be interested in whether Mondays have notably different traffic volumes than Tuesdays, and thus counts the number of cars that pass through an intersection. An analyst at an autonomous car company may notice the car behaves strangely at yellow lights, with multiple pedestrians in the crosswalk and searches for events of a yellow light, a crosswalk, and at least three pedestrians [24]. However, it is not cost effective and is too time-consuming to manually watch these growing quantities of video, so automated methods of video analysis are increasingly important in answering such queries.

Modern computer vision techniques have made great strides in automating these tasks, with near human-levels of accuracy for some tasks [36]. In particular, a commonly used approach is to perform object detection [20], which returns a sequence of bounding boxes and object class information for each frame of video, over all the frames in a video [66]. This information and simple visual features (e.g., colors) can be used to answer queries regarding the time and location of objects.

Unfortunately, there are two significant challenges in deploying these vision techniques. First, from a usability perspective, using these methods requires complex, imperative programming across many low-level libraries, such as OpenCV, Caffe2, and Detectron [28]—an often ad-hoc, tedious process. Second, from a computational perspective, the naive method of running object detection on every frame of video is infeasible at scale: state-of-the-art object detection (e.g., Mask R-CNN [35]) runs at 3 frames per second (fps), which would take 8 decades of GPU time to process 100 cameras over a month of video.

Prior work has shown that certain video queries can be highly optimized [6, 40, 46]. For example, NOSCOPE [46] and FOCUS [40] optimize the task of binary detection (presence or absence of a target class). While promising, using these pipelines face the same usability challenges as above, requiring interfacing with low-level libraries. Additionally, these pipelines are typically limited in scope (e.g., only binary detection for NOSCOPE).

To address these usability and computational challenges, we present BLAZEIT, a video query system with a declarative query language and three novel optimizations for video analytics queries not supported by prior work. To our knowledge, BLAZEIT is the first system to combine a declarative query language and an optimizer to automatically generate query-specific pipelines for video analytics at scale (as opposed to prior works such as NOSCOPE, which implement fixed-function video processing pipelines).

System Architecture. BLAZEIT consists of two components: 1) a declarative query language called FRAMEQL, and 2) a query optimization and execution engine.

BLAZEIT's first component, FRAMEQL, is its SQL-like query language, which lets users query the set of visible objects as a relation. Using a SQL-like language has two major benefits. First, as SQL is widely used, BLAZEIT can be quickly adopted by analysts and users. Second, a declarative language enables data independence, separating the specification of the system from the implementation, enabling new, video-specific query optimization. In Section 4, we demonstrate how, when combined with the standard relational algebra, FRAMEQL's schema enables a range of queries for spatiotemporal information of objects in video.

Proceedings of the VLDB Endowment, Vol. 11, No. 5
Copyright 2018 VLDB Endowment 2150-8097/18/10... $ 10.00.
DOI: https://doi.org/TBD

The second component of BLAZEIT is an end-to-end query optimizer and novel optimizations for several common classes of video queries: instead of simply running object detection to populate the rows in an FRAMEQL table, BLAZEIT leverages a range of techniques and a query optimizer for fast query execution. As BLAZEIT focuses on exploratory queries (Section 2), it lazily populates rows and examines frames as necessary. We show that using video-specific information enables several forms of query optimization beyond traditional relational databases.

Optimizations. BLAZEIT introduces three novel optimizations for common types video queries, which are not present in existing systems for querying video (Section 11):

Aggregation. First, we study the problem of optimizing aggregate queries, which can be used to provide higher-level statistics over video, (e.g., the average number of cars per hour). As in approximate query processing (AQP) [5, 37], BLAZEIT allows users to specify an error tolerance and therefore samples from the video, as opposed to performing object detection over every frame to compute the exact statistic (e.g., number of cars). However, as object detection remains the computational bottleneck, BLAZEIT further reduces the number of object detection calls by 1) rewriting queries using specialized neural networks (NNs) or 2) adapting the method of control variates [30] to video analytics. Specialized NNs [33, 46] are small NNs trained to predict the output of a larger network for a specific query. Prior work has used specialized NNs for binary classification, and in BLAZEIT we show that they can also learn to provide accurate statistics in some cases. However, in many cases, specialized NNs are not accurate enough. For these cases, we adapt the method of control variates [30], in which specialized NNs are used as a cheap to compute, correlated measure that is used to approximate an expensive statistic (i.e., object detection) to reduce sampling variance. Control variates are only beneficial when the cost of computing the control variate is significantly cheaper than the true statistic (e.g., specialized NNs vs object detection). In traditional relational DBs, the cost of materializing a row is not significantly higher than the cost of processing the row, so control variates do not reduce computation. We show that, when applied to video processing, these techniques can give up to three order-of-magnitude speedups over naive methods (e.g., using NOSCOPE to filter frames with no objects and applying object detection) and up to a 8.7× speedups over AQP.

Scrubbing. Second, we study the problem of optimizing cardinality-limited scrubbing queries [59, 60], in which a fixed number of frames that match a target predicate are returned (e.g., Tesla's autopilot is known to behave anomalously with lane dividers [53] so an analyst may look for such frames). Searching for these events sequentially or randomly is prohibitively slow when these events are rare (e.g., one event per hour). To address this problem, we adapt importance sampling from the rare-event simulation literature [45] to video analytics. Specifically, BLAZEIT uses the confidence score of a specialized NN as a proxy signal to choose which frames to perform full object detection. Intuitively, this biases the search for requested events towards sections that are more likely to contain them. We demonstrate this biased sampling can deliver up to 500× speedups over naive methods (e.g., using NOSCOPE to filter out frames with no objects and subsequently applying object detection).

Selection. Third, we study the problem of optimizing selection queries, in which users perform content-based filtering of objects (e.g., searching for red tour buses, as in Figure 1). These queries must perform object detection to obtain bounding boxes, which is computationally expensive. Thus, to reduce this computational overhead, BLAZEIT learns a set of conservative filters from the FRAMEQL query to discard irrelevant frames or parts of frames before applying object detection. We show that BLAZEIT can infer and train for classes of filters: 1) NOSCOPE's label-based filtering, 2) content-based filtering (using simple visual features), 3) temporal filtering (skipping parts of the video or subsampling frames), and 3) spatial filtering (cropping parts of the video). BLAZEIT inspects the query contents to determine which filters to apply and how to set the filter parameters. For example, if the FRAMEQL query were for red buses comprised of at least 512×512 pixels in the bottom right for at least one second, BLAZEIT can 1) filter frames for a certain redness content, 2) sample at a rate of 0.5s, and 3) crop the video to the bottom right. We demonstrate that these filters can achieve up to a 50× speedup over naive methods.

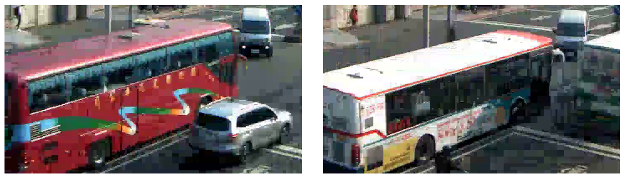
(a) Red tour bus. (b) White transit bus.

Figure 1: Examples of buses in `taipei`.

In summary, we make the following contributions:

1. We introduce FRAMEQL, a query language for spatiotemporal information of objects in videos, and show it can answer a variety of real-world queries (Section 4).
2. We introduce an aggregation algorithm that uses the method of control variates to leverage imprecise specialized NNs for more efficient aggregation than existing AQP methods (Section 6).
3. We introduce a scrubbing algorithm that leverages specialized NNs in importance sampling for finding rare events (Section 7).
4. We introduce a selection algorithm that can infer filters for discarding irrelevant frames can be inferred from FRAMEQL queries and can be applied to content-based selection for up to 50× speedups (Section 8).
5. We demonstrate that these optimizations can give up to three orders-of-magnitude speedups over naively applying object detection or NOSCOPE (Section 10).

2 Use Cases

BLAZEIT focuses on *exploratory queries*: queries that can help a user understand a video quickly, e.g., queries for aggregate statistics (e.g., number of cars) or relatively rare events (e.g., events of many birds at a feeder) in videos. We assume a large amount of archival video (i.e., the batch analytics setting) and that the full object detector has not been run over the whole video, as this would be prohibitively expensive. However, we assume that a small representative sample of the video is annotated with an object detector: this data is used as training data for filters and specialized NNs. We denote this data as the *labeled set* (which can further be split into training data and held-out data). This labeled set can be constructed once, offline, and shared for multiple queries later.

We give several scenarios where BLAZEIT could apply:

Urban planning. Given a set of traffic cameras at street corners, an urban planner performs traffic metering based on the number of cars that pass by the intersections, and determines the busiest times [71]. The planner is interested in how public transit interacts with congestion [16] and looks for clips of at least one bus and at least five cars. Then, the planner seeks to understand how tourism affects traffic and looks for red buses as a proxy for tour buses (shown in Figure 1).

Autonomous vehicle analysis. An analyst at an autonomous vehicle company notices anomalous behavior of the driving software when the car is in front of a yellow light and there are multiple pedestrians in the crosswalk [24]. Then, the analyst looks for events of a yellow light, a crosswalk, and at least three pedestrians.

Store planning. A retail store owner places a CCTV in the store [69]. The owner segments the video into aisles and counts the number of

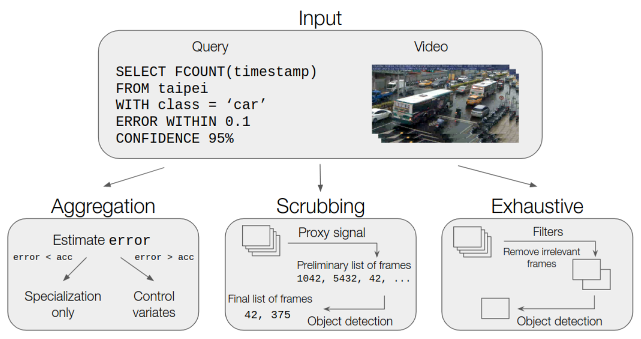

Figure 2: System diagram for BLAZEIT. BLAZEIT accepts FRAMEQL queries and chooses a query plan depending on the type of query.

people that walk through each aisle to understand which products are popular and which ones are not. This information can be used for planning store layout, aisle layout, and product placement.

Ornithology. An ornithologist (a scientist who studies birds) may be interested in understanding bird feeding patterns. The ornithologist might place a webcam in front of a bird feeder [1]. Then, the ornithologist might put different bird feed on the left and right side of the feeder. Finally, the ornithologist can count the number of birds that visit the left and right side of the feeder. As a proxy for species, the ornithologist might then select red or blue birds.

These queries can be answered using spatiotemporal information of objects in the video, along with simple functions over the content of the boxes. Thus, these applications illustrate a need for a unified method of expressing such queries.

3 BLAZEIT System Overview

In this section, we give a brief overview of BLAZEIT's system architecture: BLAZEIT's query language, FRAMEQL, its query optimizer, and its execution engine.

FRAMEQL allows users to express queries for spatiotemporal information of objects in video (further details in Section 4). The key challenge we address in this work is efficient execution of FRAMEQL queries: while performing object detection, entity resolution, and UDFs over each frame can answer FRAMEQL queries, this procedure is prohibitively slow. Therefore, we present optimizations for three common classes of queries: aggregation (Section 6), scrubbing (Section 7), and content-based selection (Section 8). Implementation details are given in Section 9. A system diagram of BLAZEIT is shown in Figure 2.

3.1 Components

Configuration. As user needs are distinct, BLAZEIT contains several configurable components: the object detection method, the entity resolution method (resolving object across frames), and optional UDFs. While we provide defaults, these components can be changed, e.g., a license plate reader could be used for resolving the identity of cars. The UDFs can be used to answer more complex queries, such as determining color, filtering by object size or location, or fine-grained classification. UDFs are functions that accept a timestamp, mask, and rectangular set of pixels. As an example, to compute the "redness" of an object, the UDF could use OpenCV to average the red channel of the pixels.

Filters. Many filters BLAZEIT uses are statistical in nature, so the optimizer must account for their error rates (Section 8). For example, BLAZEIT will train a specialized NN to filter for frames with buses for content-based selection for red buses. However, the specialized NN may not be accurate on every frame. To account for this error rate, BLAZEIT uses a held-out set of frames to estimate the selectivity and error rate. Given an error budget, BLAZEIT's query optimizer selects between the filters and uses rule-based optimization to select the fastest query plan. Finally, BLAZEIT can always ensure no false positives by running the most accurate method on the relevant frames.

Specialized neural networks. Throughout, we use specialized NNs as a core primitive [46] (we use a miniaturized ResNet [36] in this work as described in Section 9, but other network architectures could also be used). A specialized NN is a neural network that has been trained to mimic a larger NN (e.g., Mask R-CNN) on a simplified task, e.g., on a marginal distribution of the larger NN. For example, NOSCOPE's simplified task is the binary detection task. As the specialized NN is predicting a simpler output, it can run dramatically faster. Prior work has specialized NNs for binary detection [33,46], but we extend specialization to count and perform multi-class classification. Additionally, we apply various statistical methods over the results of specialized NNs to accurately answer queries such as aggregation or scrubbing.

Bootstrapping filters. To bootstrap and train the filters and specialized NNs, we assume the presence of a labeled set: a small, representative sample that the object detector was run over. Notably, this procedure can be done automatically.

3.2 Limitations

While BLAZEIT can answer significantly more classes of video queries than prior work, we highlight several limitations.

Model Drift. Our current implementation assumes the labeled set for the filters obtained in BLAZEIT's optimizer is from the same distribution as the remaining video to be analyzed. If the distribution changes dramatically in the new video (e.g., a sunny day vs an extremely foggy day), BLAZEIT will need to re-train the filters. To our knowledge, tracking model drift in visual data has not been well characterized and existing systems such as NOSCOPE [46] do not handle model drift. Thus we view the automatic detection and mitigation of model drift as an exciting area of future research.

Labeled set. Currently, BLAZEIT requires the object detection method to be run over a portion of the data to obtain data for training specialized NNs and filters. We view the problem of warm-starting filters and specialized NNs as an exciting area of future work.

Object detection. BLAZEIT is dependent on the user-defined object detection method and does not support object classes outside what the method returns. For example, the pretrained version of Mask R-CNN [28,35] can detect cars, but cannot distinguish between sedans and SUVs. However, users could supply UDFs for these queries.

Scale-out. Our BLAZEIT prototype is written for a single server, namely we do not address scale-out. However, as the majority of BLAZEIT's operators are data-parallel, subsets of the video can be processed on different servers, which could be implemented on a system such as SCANNER [65].

Actions and events. BLAZEIT currently does not support action queries (e.g., person crossing the street) or events (e.g., car nearly hitting a bike). However, to our knowledge, these queries are difficult to answer accurately even with state-of-the-art computer vision methods [47]. As computer vision advances, we view the problem of extending query support as an exciting area of future work.

4 FrameQL: A Query Language for Complex Visual Queries over Video

To address the need for a unifying query language over video analytics, we introduce FRAMEQL, a SQL-like language for querying spatiotemporal information of objects in video. We choose a declarative language

Field	Type	Description
`timestamp`	float	Time stamp
`class`	string	Object class (e.g., bus, car, person)
`mask`	(float, float)*	Polygon containing the object of interest, typically a rectangle
`trackid`	int	Unique identifier for a continuous time segment when the object is visible
`content`	float*	Content of pixels in mask
`features`	float*	The feature vector output by the object detection method.

Table 1: FRAMEQL's data schema contains spatiotemporal and content information related to objects of interest, as well as metadata (class, identifiers). Each record represents an object appearing in one frame; thus a frame may have many or no records. The features can be used for downstream tasks.

for two reasons. First, encoding queries via a declarative language interface separates the specification and implementation of the system, which enables query optimization (discussed later). Second, as SQL is the lingua franca of data analytics, FRAMEQL can be easily learned by users familiar with SQL and enables interoperability with relational algebra.

FRAMEQL allows users to query the frame-level contents of a given video feed, specifically the objects appearing in the video over space and time by content and location. FRAMEQL represents videos (stored and possibly compressed in formats such as H.264) as relations, with one relation per video. As in SQL, FRAMEQL allows selection, projection, and aggregation of objects, and, by returning relations, can be composed with standard relational operators. By providing a table-like schema using the standard relational algebra, we enable users with only familiarity with SQL to query videos, whereas implementing these queries manually would require expertise in deep learning, computer vision, and programming.

We show FRAMEQL's data schema in Table 1. FRAMEQL's data schema contains fields relating to the time, location, and class of objects, scene and global identifiers, the box contents, and the features from the object detection method (described below). While BLAZEIT provides a default method of populating the schema, the user can specify the object detection method (which will populate `mask`, `class`, and `features`) and the entity resolution method (which will populate `trackid`). For example, an ornithologist may use an object detector that can detect different species of birds, but an autonomous vehicle analyst may not need to detect birds at all. Given these methods, BLAZEIT can automatically populate the data schema. BLAZEIT aims to be as accurate as the configured methods, specifically BLAZEIT does not aim to be more accurate than the configured methods.

Prior visual query engines have proposed similar schemas, *but assume that the schema is already populated* [48, 54], i.e. that the data has been created through external means (typically by humans). In contrast, FRAMEQL's schema can be automatically populated by BLAZEIT. However, as we focus on exploratory queries in this work, FRAMEQL's schema is *virtual* and rows are only populated as necessary for the query at hand. This is similar to an unmaterialized view. This form of laziness enables a variety of optimizations via query planning.

FRAMEQL's schema. We briefly describe the fields in FRAMEQL. Recall that a record corresponds to an object that appears in a frame.

- `timestamp` is the timestamp of the object. There is a one-to-one correspondence between timestamps and frames of the video.
- `class` is the object type (e.g., car or bus). The specificity of `class` is determined by the object detection method (e.g., Mask R-CNN on MS-COCO does not support sedan vs SUV).
- `mask` is the polygon that encloses the object. We only consider `mask` in the form of bounding boxes, but semantic segmentation [27] could be used for finer grained masks.

```
SELECT * | expression [, ...]
  FROM table_name
  [ WHERE condition ]
  [ GROUP BY expression [, ...] ]
  [ HAVING condition [, ...] ]
  [ LIMIT count ]
  [ GAP count ]
  [ ERROR WITHIN tolerance AT CONFIDENCE conf ]
```

Figure 3: FRAMEQL's syntax.

Syntactic element	Description
`FCOUNT`	Frame-averaged count (equivalent to time-averaged count), i.e., `COUNT(*) / MAX(timestamp)`
`ERROR WITHIN`	Absolute error tolerance
`FPR WITHIN`	Allowed false positive rate
`FNR WITHIN`	Allowed false negative rate
`CONFIDENCE`	Confidence interval
`GAP`	Minimum distance between returned frames

Table 2: Additional syntactic elements in FRAMEQL. Some of these were adopted from BlinkDB.

- `trackid` is a unique identifier for the object as it is visible through a continuous segment in time. If the object exists and re-enters the scene, it will be assigned a new `trackid`.
- `content` is the pixels in as contained by `mask`.
- `features` are the features from the object detection method. The features can be used for downstream tasks, such as fine-grained classification.

FRAMEQL's query format. We present the syntax for FRAMEQL in Figure 3. FRAMEQL contains a subset of standard SQL with three extensions: `GAP`, syntax for specifying an error tolerance, and `FCOUNT`. Notably, as we do not optimize joins in this work, we do not support joins (but we describe how to extend FRAMEQL with joins in Appendix A.4). We show FRAMEQL's extensions Table 2; several were taken from BlinkDB [5]. We briefly provide the motivation behind each additional piece of syntax.

First, when the user selects timestamps, the `GAP` keyword provides a way to ensure that the returned frames are at least `GAP` frames apart. For example, if 10 consecutive frames contains a car and `GAP` $=100$, only one frame of the 10 would be returned.

Second, as in BlinkDB [5], users may wish to have fast response to exploratory queries and may tolerate some error. Thus, we allow the user to specify error bounds in the form of absolute error, false positive rate, and false negative rate, along with a specified confidence (see below for examples). NOSCOPE's pipeline can be replicated with FRAMEQL using these constructs. We choose absolute error bounds in this work, as they allow for adaptive sampling procedures (Section 6).

Finaly, as we provide absolute error bounds, we provide a short-hand for returning a frame-averaged count, which we denote as `FCOUNT`. For example, consider two videos: 1) a 10,000 frame video with one car in every frame, 2) a 10 frame video with a car only in the first frame. Then, `FCOUNT` for the average number of cars per frame would return 1 in the first video and 0.1 in the second video. As videos vary in length, this allows for a normalized way of computing errors. `FCOUNT` can easily be transformed into a time-averaged count.

FRAMEQL examples. We first describe how the some of the example use cases from Section 2 can be written in FRAMEQL. In the following examples, we assume the video is recorded at 30 fps.

Figure 4a shows how to count the average number of cars in a frame. The query uses `FCOUNT` as the error bounds are computed per-frame.

Figure 4b shows how to select frames with at least one bus and at least five cars. This query uses the `GAP` keyword that ensures events are a certain time apart. At 30 fps, `GAP 300` corresponds to 10 seconds.

```
SELECT FCOUNT(*)
FROM taipei
WHERE class = 'car'
ERROR WITHIN 0.1
AT CONFIDENCE 95%
```

(a) The FRAMEQL query for counting the frame-averaged number of cars within a specified error and confidence.

```
SELECT timestamp
FROM taipei
GROUP BY timestamp
HAVING SUM(class='bus')>=1
   AND SUM(class='car')>=5
LIMIT 10 GAP 300
```

(b) The FRAMEQL query for selecting 10 frames of at least one bus and five cars, with each frame at least 10 seconds apart (at 30 fps, 300 frames corresponds to 10s).

```
SELECT *
FROM taipei
WHERE class = 'bus'
  AND redness(content) >= 17.5
  AND area(mask) > 100000
GROUP BY trackid
HAVING COUNT(*) > 15
```

(c) The FRAMEQL query for selecting all the information of red buses at least 100,000 pixels large, in the scene for at least 0.5s (at 30 fps, 0.5s is 15 frames). The last constraint is for noise reduction.

Figure 4: Three FRAMEQL example queries.

Figure 4c shows how to exhaustively select frames with red buses. Here, `redness` and `area` are UDFs, as described in Section 3. Here, 15 frames corresponds to 0.5 seconds.

The other example use cases can be answered in a similar manner. We give further examples of FRAMEQL's syntax in Appendix A.3.

5 BLAZEIT Query Execution Overview

When the user issues an FRAMEQL query, BLAZEIT's query engine optimizes and executes the query. BLAZEIT's primary challenge is executing the query *efficiently*: naive methods, such as performing object detection on every frame or using NOSCOPE [46] as a filter, are often prohibitively slow. To optimize and execute the query, BLAZEIT inspects the query contents to see if optimizations can be applied. For example, BLAZEIT cannot optimize `SELECT *`, but can optimize aggregation queries with a user-specified error tolerance. Currently, BLAZEIT uses a rule-based optimizer.

Because object detection is the major computational bottleneck, BLAZEIT's optimizer primarily attempts to reduce the number of object detection calls while achieving the target accuracy. As object detection methods have increased in accuracy, they have similarly increased in computational complexity. For example, YOLOv2 [66] runs at approximately 80 fps, with an mAP score of 25.4 on MS-COCO [55] (mAP is an object detection metric, with values between 0 and 100, higher being better), but the most accurate version of Mask R-CNN [35] provided by the Detectron framework [28] run at 3 fps with a mAP of 45.2. As a result, object detection is, by far, the most computationally expensive part of a FRAMEQL query; for reference, the specialized NNs we use in this work run at 10,000 fps and some of our simple filters run at 100,000 fps.

BLAZEIT leverages three novel optimizations and existing techniques from NOSCOPE to reduce the computational cost of object detection, targeting aggregation (Section 6), scrubbing (Section 7), and content-based selection (Section 8). As the filters and specialized NNs we consider are cheap compared to the object detection methods, they are almost always worth calling: a filter that runs at 100,000 fps would need to filter 0.003% of the frames to be effective. Thus, we have found a rule-based optimizer to be sufficient in optimizing FRAMEQL queries.

We describe BLAZEIT's novel optimizations in turn. For each optimization, we describe the overview, the physical operator(s), its time complexity and correctness, and the operator selection procedure.

```
Data: Training data, held-out data, unseen video, $uerr \leftarrow$ user's
      requested error rate, $conf \leftarrow$ user's confidence level
Result: Estimate of requested quantity
if training data has instances of object then
    train specialized NN on training data;
    $err \leftarrow$ specialized NN error rate on held-out data;
    $\tau \leftarrow$ average of specialized NN over unseen video;
    if $P(err < uerr) < conf$ then
        return $\tau$;
    else
        $\hat{m} \leftarrow$ result of Equation 2 (control variates);
        return $\hat{m}$;
    end
else
    Return result of traditional AQP;
end
```

Algorithm 1: BLAZEIT's procedure to run an aggregation query.

6 Optimizing Aggregates

Overview. In an aggregation query, the user is interested in some statistic over the data, such as the average number of cars per frame; an example is given in Figure 4a. To exactly answer these queries, BLAZEIT must call object detection on every frame, which is prohibitively slow. However, if the user specifies an error tolerance (Section 4), BLAZEIT can leverage a range of optimizations for accelerated query execution.

In this work, we focus on optimizing counting the number of objects in a frame and describe other cases in the appendix. BLAZEIT requires training data (Section 2) of the desired quantity (e.g., number of cars) to leverage specialized NNs. If there is not sufficient training data, BLAZEIT will default to traditional AQP. If there is sufficient training data, BLAZEIT will first train a specialized NN to estimate the statistic per frame: if the specialized NN is accurate enough, then BLAZEIT can return the answer directly. Otherwise, BLAZEIT will use the method of control variates, which uses specialized NNs to reduce the variance of AQP, thus requiring fewer samples. We next describe the steps in detail.

Operator Selection. The intuition above is formalized in Algorithm 1. BLAZEIT first determines whether there is sufficient training data ($> 1\%$ of the data has instances of the object) to train a specialized NN. In cases where the training data does not contain enough examples of interest (e.g., a video of a street intersection is unlikely to have examples of bears), BLAZEIT will default to standard AQP. We use a slightly modified adaptive sampling algorithm that respects the user's error bound, which is inspired by Online Aggregation [37] and BlinkDB [5].

When there is sufficient training data, BLAZEIT will train a specialized NN and estimate its error rate on a held-out set. If the error is within the user-specified error and confidence level, it will then execute the specialized NN on the unseen data (subsampled in half) and return the answer directly, foregoing the object detection method entirely. As specialized NNs are significantly faster than object detection, this results in much faster execution.

When the specialized NN is not accurate enough, it is used as a control variate: an auxiliary variable that is cheap to compute but highly correlated with the true statistic. Control variates lets us approximate the answer with fewer samples compared to random sampling.

Physical Operators. We describe the procedures for sampling, query rewriting, and control variates below.

Sampling. When the query contains a tolerated error rate and there is not sufficient training data for a specialized NN, BLAZEIT can sample from the video and only populate a small number of rows (or not populate them at all) for faster execution. Similar to online aggregation [37], we provide absolute error bounds. However, we present a sampling procedure that terminates based on the sampling variance and a CLT bound [56]. Terminating based on sampling variance allows variance reduction methods to terminate early, which is useful for control variates.

For an absolute error bound of ϵ (irrespective of the confidence level), we require at least $\frac{K}{\epsilon}$ samples, where K is the range of the estimated quantity (derived from an ϵ-net argument [34]). For example, if the user queries for the average number of cars per frame, K would be the maximum number of cars over all frames plus one.

Thus, in BLAZEIT's adaptive sampling procedure, we being with $\frac{K}{\epsilon}$ samples. At step n, we increase the number of samples by $\frac{nK}{\epsilon}$. We terminate when the CLT bound gives that the error rate is satisfied at the given confidence level, namely,

$$Q(1-\frac{\delta}{2}) \cdot \hat{\sigma_N} < \epsilon \tag{1}$$

where δ is the confidence interval, $\hat{\sigma_N}$ is the sample standard deviation at round N, and Q is the percent point function (i.e. the inverse of the cumulative distribution function) for the normal distribution [37]. We use the finite sample correction to compute the sample standard deviation.

Specialized Networks for Query Rewriting. In cases where the specialized NN is accurate enough (as determined by the bootstrap on the held-out set; the accuracy of the specialized NN depends on the noisiness of the video and object detection method), BLAZEIT can return the answer directly from the specialized NN run over all the frames for dramatically faster execution and bypass the object detection entirely. BLAZEIT uses multi-class classification for specialized NNs to count the number of objects in a frame.

To train the specialized NN, BLAZEIT selects the number of classes equal to the highest count that is at least 1% of the video plus one. For example, if 1% of the video contains 3 cars, BLAZEIT will train a specialized NN with 4 classes, corresponding to 0, 1, 2, and 3 cars in a frame. BLAZEIT uses 150,000 frames for training and uses a standard training procedure for NNs (SGD with momentum [36]) for one epoch.

BLAZEIT estimates the error of the specialized NN on a held-out set using the bootstrap [19]. If the error is low enough at the given confidence level, BLAZEIT will process the unseen data using the specialized NN and return the result.

Control Variates. In cases where the user has a stringent error tolerance, specialized NNs may not be accurate enough to answer a query on their own. To reduce the cost of sampling from the object detector, BLAZEIT introduces a novel method to take advantage of specialized NNs while still achieving high accuracy, by combining specialized NNs with AQP-like sampling. In particular, we adopt the method of control variates [30] to video analytics (to our knowledge, control variates have not been applied to database query optimization or video analytics). Specifically, control variates is a method of variance reduction (variance reduction is a standard technique in Monte Carlo sampling [68] and stochastic optimization [44]) which uses a proxy variable correlated with the statistic of interest. Intuitively, by reducing the variance of sampling, we can reduce the number of frames that have to be sampled and processed by the full object detector.

To formalize this intuition, suppose we wish to estimate the expectation of a quantity m and we have access to an auxiliary variable t. The desiderata for t are that: 1) t is cheaply computable, 2) t is correlated with m (see time complexity). We further assume we can compute $\mathbb{E}[t] = \tau$ and $Var(t)$ exactly. Then,

$$\hat{m} = m + c(t - \tau) \tag{2}$$

is an unbiased estimator of m for any choice of c [30]. The optimal choice of c is $c = -\frac{Cov(m,t)}{Var(t)}$ and using this choice of c gives $Var(\hat{m}) = (1 - Corr(m,t)^2)Var(m)$. As an example, suppose $t = m$. Then, $\hat{m} = m + c(m - \mathbb{E}[m]) = \mathbb{E}[m]$ and $Var(\hat{m}) = 0$.

This formulation works for any choice of t, but choices where t is correlated with m give the best results. As we demonstrate in Section 10.2, specialized networks can provide a correlated signal to the ground-truth object detection method for all queries we consider.

As an example, suppose we wish to count the number of cars per frame. Then, m is the random variable denoting the number of cars the object detection method returns. In BLAZEIT, we train a specialized NN to count the number of cars per frame. Ideally, the specialized model would exactly mimic the object detection counts, but this is typically not the case. However, the specialized NNs are typically correlated with the true counts. Thus, the random variable t would be the output of the specialized NN. As our choice of specialized NNs are extremely cheap to compute, we can calculate their mean and variance exactly on all the frames. BLAZEIT estimates $Cov(m,t)$ at every round.

Correctness. BLAZEIT's sampling-based methods will return approximate answers that respect the requested error bound and confidence level. We defer the proofs to the appendix.

Query rewriting using specialized NNs will respect the requested error bound and confidence level under the assumption of no model drift (see Section 3.2).

Time and sample complexity. BLAZEIT must take $c_\delta \frac{\sigma^2}{\epsilon^2}$ samples from a random variable with standard deviation σ (c_δ is a constant that depends on the confidence interval, see Equation 1). Denote the standard deviation of AQP as σ_a and from control variates as σ_c; the amortized cost of running a specialized NN on a single frame as k_s and of the object detection method as k_o; the total number of frames as F.

Control variates are beneficial when $k_s F < k_o \frac{c_\delta}{\epsilon^2}(\sigma_a^2 - \sigma_c^2)$. Thus, as the error bound decreases or the difference in variances decreases (which typically happens when specialized NNs are highly accurate or when σ_a is large), control variates become more beneficial.

While σ_a and σ_c depend on the query, we empirically show in Section 10 and Appendix C when control variates and query rewriting are beneficial.

7 Optimizing Scrubbing Queries

Overview. In cardinality-limited scrubbing queries, the user is interested in finding rare events (e.g., a bus and five cars as shown in Figure 4b), as if the event is common, the user can simply watch the video. To answer this query, BLAZEIT could run the object detection method over every frame to search for the event. However, if the event occurs infrequently, naive methods of random sampling or sequentially processing the video can be prohibitively slow (e.g., at a frame rate of 30 fps, an event that occurs, on average, once every 30 minutes corresponds to a rate of 1.9×10^{-5}).

Our key intuition is to bias the search towards regions of the video that likely contain the event. To bias the search, we use specialized NNs, and combine them with techniques from the rare-event simulation literature [45]. As an example of rare-event simulation, consider the probability of flipping 80 heads out of 100 coin flips. Using a fair coin, the probability of encountering this event is astronomically low (rate of 5.6×10^{-10}), but using a biased coin with $p = 0.8$ can be orders of magnitude more efficient (rate of 1.2×10^{-4}) [45].

Physical operator and selection. BLAZEIT currently supports scrubbing queries searching for at least N of an object class (e.g., at least one bus and at least five cars). In BLAZEIT, we use specialized NNs to bias which frames to sample. The overall procedure is as follows:

- If there are examples no instances of the query in the training set, BLAZEIT will default to performing the object detection method over every frame and applying applicable filters as described in Section 8 (random sampling is also possible).
- If there are examples, BLAZEIT will train a specialized NN to recognize frames that satisfy the query.

- BlazeIt rank orders the unseen data by the confidence that the frame matches the predicate from the specialized NN.
- BlazeIt will perform object detection in the rank order until the requested number of events is found.

For a given query, BlazeIt trains a specialized NN to recognize frames that satisfy the query. While we could train a specialized NN as a binary classifier of the frames that satisfy the predicate and that do not, we have found that rare queries have extreme class imbalance. Thus, we train the specialized NN to count instead, which alleviates the class imbalance issue; this procedure has the additional benefit of allowing the trained specialized NN to be reused for other queries such as aggregation. For example, suppose the user wants to find frames with at least one bus and at least five cars. Then, BlazeIt trains a single specialized NN to separately count buses and cars. The signal BlazeIt uses is the sum of the probability of the frame having at least one bus and at least five cars. BlazeIt takes the most confident frames until the requested number of frames is found.

In the case of multiple object classes, BlazeIt trains a single NN to detect each object class separately (e.g., instead of jointly detecting "car" and "bus", the specialized NN would return a separate confidence for "car" and "bus"), as this results in fewer weights and typically higher performance.

After the results are sorted, the full object detector is applied until the requested number of events is found or all the frames are searched. If the query contains `GAP`, once an event is found, the surrounding `GAP` frames are ignored.

Correctness. In scrubbing queries, a correct answer is always returned, as the object detection method is called on all sampled frames. All frames will be exhaustively searched if there are fewer events than the number requested.

Time complexity. Denote K to be the number of events the user requested, N the total number of matching events, and F the total number of frames in the video. We denote, for event i, f_i as the frame where the event occurred. Throughout, once an event is found, the `GAP` frames around the event can be ignored, but as this is typically inconsequential to the runtime in practice, we ignore it in the analysis.

If $K > N$, then every method must consider every frame in the video, i.e., F frames. From here on, we assume $K \leq N$.

For sequential scans, f_K frames must be examined.

For random sampling, consider the number of frames to find a single event. In expectation, random sampling will consider $\frac{F}{N}$ frames. Under the assumption that $K \ll N \ll F$, then random sampling will consider approximately $\frac{K \cdot F}{N}$ frames (in expectation).

While using specialized NNs to bias the search does not guarantee faster runtime, we show in Section 10 that it empirically can reduce the number of frames considered.

8 Optimizing Content-based Selection

Overview. In a content-based selection, the user is interested information about the mask or content of every instance of an event, e.g., finding red buses (Figure 4c). In these queries, the object detection method must be called to obtain the mask. As object detection is the overwhelming computational bottleneck, BlazeIt aims to perform object detection as few times as possible. For example, BlazeIt can filter frames that lack red *before* performing object detection to look for buses.

To reduce the number of object detection invocations, BlazeIt infers filters to discard frames irrelevant to the query *before* running object detection on them. BlazeIt currently supports four classes of filters: 1) label-based filtering, 2) content-based filtering, 3) temporal filtering, and 4) spatial filtering (described in detail below). Importantly, these filter types and parameters are automatically selected from the query and training data as described below.

While some filters can be applied with no false positives, others filters are statistical in nature and may have some error rate. The error rate of these filters can be estimated on a held-out set, as in cross-validation [25]. However, as prior work, such as NoScope [46], has considered how to set these error rates, we only consider the case where the filters are set to have no false negatives on the held-out set. Assuming the held-out set is representative of the unseen data (i.e., no model drift, see Section 3.2), this will incur few false negatives on the unseen data, which is sufficient for exploratory queries or `LIMIT` queries.

Physical Operators. In BlazeIt, we present instantiations of each class of filter to demonstrate their effectiveness. We describe each class of filter and BlazeIt's instantiations of the filter class.

Label-based filtering. In label-based filtering, the video is filtered based on the desired labels. We leverage similar techniques to NoScope [46], namely training a specialized NN to discard frames without the label.

Content-based filtering. In content-based filtering, the video is filtered based on UDFs that return continuous values (e.g., the UDF `redness` returns a measure of redness of an image). BlazeIt will attempt to learn a threshold based on the training data or fall back to naive application of the UDF if it cannot learn a beneficial threshold.

Currently, BlazeIt supports automatically learning a filter from any UDF that only takes the box content, by applying the UDF over the entire frame. For example, if the query filters for `redness > R` and `area > A`, then `redness(frame) > A · R` is a safe threshold. In practice, even more conservative thresholds may work depending on the UDFs and video.

We describe when content-based filtering is effective below.

Temporal filtering. In temporal filtering, the video is filtered based on temporal cues. For example, the analyst may want to find buses in the scene for at least K frames. In this case, BlazeIt subsamples the video at a rate of $\frac{K-1}{2}$. BlazeIt additionally support basic forms of filtering such as "query the video from 10AM to 11AM."

Spatial filtering. In spatial filtering, only regions of interest (ROIs) of the scene are considered. For example, a street may have cars parked on the side but the analyst may only be interested in vehicles in transit, so the analyst specifies in the query which parts of the scene contain moving vehicles. The ROI is specified by the user and can be used in smaller models for faster inference, and activity outside the ROI can be ignored, which can increase the selectivity of other filters.

BlazeIt crops the images to be more square if the given ROI allows such an operation. For example, if the query only looks for objects with `xmax(mask) < 720` in a 1280×720 video, BlazeIt will resize the frames to be 720×720. Standard object detectors run faster when the input is more square: in most existing detectors, the input image is resized so that the short-edge is a specific size and the aspect ratio is held constant [35, 67] (for a fixed short-edge size, reducing the long-edge size will make the image smaller). As the computation scales with the resolution, square images result in the least computation.

Operator Selection. BlazeIt will infer which filters can be applied from the user's query. We describe how each class of filter can be inferred from the query.

First, if the user selects an area of the video, BlazeIt resizes the frame to be as square as possible, while keeping the area (along with some padding) visible, as described above.

Second, BlazeIt infers the times in the video and the subsampling rate from the query to achieve exact results. For example, if the user queries for objects present in the video for at least 30 frames (1 second), BlazeIt can sample once very 14 frames.

Third, if the user selects a class or set of classes, BlazeIt trains a specialized NN to detect these classes, as in NoScope. Then, BlazeIt estimates the threshold on unseen data to ensure no false negatives.

Fourth, if the user provides a content-based UDF over the content (e.g., determining the color of the object), BLAZEIT can apply the UDF over the entire frame (as opposed to the box), and filter frames that do not satisfy the UDF at the frame level. BLAZEIT sets UDF filter thresholds similar to how it sets thresholds for specialized NNs. However, for this procedure to be effective, the UDF must return a continuous value that can be scaled to a confidence. Consider two possible UDFs for redness: 1) a UDF which returns true if the over 80% of the pixels have a red-channel value of at least 200 (out of 256) and 2) a UDF that returns the average of the red-channel values. While both UDFs can be used as a filter, the second will be vastly more effective.

Correctness. BLAZEIT will perform object detection on all frames that pass its filters, so no false positives will be returned. Spatial and temporal filters are exact, but label-based and content-based filters are not. For the probablistic filters, we set the thresholds so there are no false negatives on the held-out set. The accuracy will be high, but possibly not perfect, and will return no false positives for `LIMIT` queries.

Time complexity. Before filter i is applied, denote the remaining frames F_i. Denote the cost of filter i on a frame to be c_i and the cost of object detection to be c_o. Then, if $c_i \cdot F_i < c_o \cdot (F_i - F_{i+1})$, then the filter is beneficial to run. The filters we consider in this work are 3 to 6 orders of magnitude faster than object detection and are thus nearly always worth running.

9 Implementation

We implement an open-source[1] BLAZEIT prototype that implements the above query optimizer. We implement our prototype in Python 3.5 as the deep learning frameworks we use for object detection require Python. To interface with these libraries, we implement the control plane in Python. For efficiency purposes, we implement the non-NN filters in C++. We use PyTorch v0.4 for the training and evaluation of specialized models. For object detection, we use FGFA [75] using MXNet v1.2 and Mask R-CNN [35] using the Detectron framework [28] in Caffe v0.8. We modify the implementations to accept arbitrary parts of video. For FGFA, we use the provided pre-trained weights and for Mask R-CNN, we use the pretrained `X-152-32x8d-FPN-IN5k` weights. We ingest video via OpenCV.

Currently, BLAZEIT uses a rule-based optimizer for query rewriting [64], which supports the queries in Section 4. Most queries follow the following general steps: 1) train specialized neural networks and filters for the query at hand, 2) compute statistics on a held-out dataset to estimate the error or selectivity of the NNs and filters, 3) choose a plan for the unseen data, 4) execute the plan.

BLAZEIT uses a fluent DSL written in Python to specify queries. The materialized data is stored in Pandas dataframes (as storing and processing the materialized data is not a significant portion of the runtime).

We briefly overview different parts of the implementation.

Video ingestion. BLAZEIT initially loads the video and resizes the frames to the appropriate size for each model (65×65 for specialized NNs, short side of 600 pixels for object detection methods), and normalizes the pixel values appropriately. Additionally, we can preprocess the video and directly store the result for faster ingestion.

Specialized NN training. We train the specialized NNs using PyTorch v0.4. Video are ingested and resized to 65×65 pixels and normalized using standard ImageNet normalization [36]. Standard cross-entropy loss is used for training, with a batch size of 16. We used SGD with a momentum of 0.9. Our specialized NNs use a "tiny ResNet" architecture, a modified version of the standard ResNet architecture [36], which has 10 layers and a starting filter size of 16, for all query types. As this work focuses on exploratory queries, we chose tiny ResNet as a good default and show that it generally performs better than the models used in [46] in Appendix C.

Identifying objects across frames. Our default implementation for computing `trackid` use motion IOU [75], but is configurable. Given the set of objects in two consecutive frames, we compute the pairwise IOU of each object in the two frames. We use a cutoff of 0.7 to call an object the same across consecutive frames.

10 Evaluation

We evaluate BLAZEIT on a variety of FRAMEQL queries on real-world video streams in three scenarios: 1) aggregate queries, 2) scrubbing queries for rare events, and 3) accurate, spatiotemporal queries over a variety of object classes. We illustrate that:

1. BLAZEIT achieves up to 4000× increased throughput compared to a naive baseline, up to a 2500× speedup compared to NOSCOPE, and up to a 8.7× speedup over AQP (Section 10.2) on aggregation queries.
2. BLAZEIT achieves up to 1000× speedup compared to a naive baseline and a 500× speedup compared to NOSCOPE for video scrubbing queries (Section 10.3).
3. BLAZEIT achieves up to 50× speedup for content-based selection over naive methods by automatically inferring filters to apply before object detection (Section 10.4).

10.1 Experimental Setup

Evaluation queries and videos. We evaluate BLAZEIT on six videos shown in Table 3, which were scraped from YouTube. `taipei`, `night-street`, `amsterdam`, and `archie` were from the same cameras as in NOSCOPE (the other streams were removed from YouTube, so we were unable to use them) and we collected two other streams. We only consider times where the object detection method can perform well (due to lighting conditions), which resulted in 6-11 hours of video per day. These datasets vary in object class (car, bus, boat), occupancy (12% to 90%), and average duration of object appearances (1.4s to 10.7s). For each webcam, we use three days of video: one day for training labels, one day for threshold computation, and one day for testing (as in [46]).

We evaluate on queries similar to Figure 4, in which the class and video were changed.

Choice of object detection method. We label a portion of each video using Mask R-CNN [35], FGFA [75], and YOLOv2 [66], and manually select the object detection that was the most accurate for each video. As object detection methods have improved since NOSCOPE was published, we did not select YOLOv2 for any of the videos.

In timing the naive baseline, we only included the GPU compute time and excluded the time to process the video and convert records to FRAMEQL format, object detection is the overwhelming computational cost (see Appendix A.1).

Data preprocessing. The literature reports that state-of-the-art object detection methods still suffer in performance for small objects [35,75], which we empirically observe even for newer detectors. Thus, we only consider regions where objects are large relative to the size of the frame (these regions are video dependent). Object detectors will return a set of boxes and confidences values. We manually select confidence thresholds for each video and object class for when to consider an object present, shown in Table 3.

Evaluation metrics. We compute all accuracy measures with respect to the object detection method, in which we treat the object detection method as ground truth. For aggregate statistical queries, we report the absolute error. For scrubbing queries, we guarantee only true positives are returned, thus we only report throughput. Finally, for queries that

[1] `https://github.com/stanford-futuredata/blazeit`

Video Name	Object	Occupancy	Average duration	Distinct count	Resol.	FPS	# Eval frames	Length (hrs)	Object detection method	Thresh
taipei	bus	11.9%	2.82s	1749	720p	30	1188k	33	FGFA	0.2
	car	64.4%	1.43s	32367						
night-street	car	28.1%	3.94s	3191	720p	30	973k	27	Mask R-CNN	0.8
rialto	boat	89.9%	10.7s	5969	720p	30	866k	24	Mask R-CNN	0.8
grand-canal	boat	57.7%	9.50s	1849	1080p	60	1300k	18	Mask R-CNN	0.8
amsterdam	car	44.7%	7.88s	3096	720p	30	1188k	33	Mask R-CNN	0.8
archie	car	51.8%	0.30s	90088	2160p	30	1188k	33	Mask R-CNN	0.8

Table 3: Video streams and object labels queried in our evaluation. While there are three days of video total for each stream, we show the data from the test set, as the data from the test set will influence the runtime of the baselines and BLAZEIT.

require detailed information about object (i.e. queries that require performing object detection), all our errors are false negatives, because every frame chosen by our methods is passed to the object detector. Thus, we report the false negative rate for these queries.

In this work, we consider accuracy at the *frame-level*, as we have empirically found that modern object detection methods can return frame-level accurate results. This is in contrast to to the one-second binning that is used in [46] to mitigate label flickering for NOSCOPE.

We measure throughput by timing the complete end-to-end system excluding the time taken to decode video, as is standard [46]. We additionally assume the labeled set is computed offline once, so we exclude the time to generate the labeled set (as in [46], we currently use a day of video for training and a day of video for the held-out set). Unlike in [46], we also show runtime numbers *when the training time of the specialized model is included* (excluded in [46]). We include this time as BLAZEIT focuses on exploratory queries, whereas NOSCOPE focuses on long-running streams of data. We additionally show numbers where the training time is excluded, which could be achieved if the specialized NNs were indexed ahead of time.

Hardware Environment. We perform our experiments on a server with an NVIDIA Tesla P100 GPU and two Intel Xeon E5-2690v4 CPUs (56 threads). The system has a total of 504 GB of RAM.

10.1.1 NoScope Baseline Configuration

To our knowledge, NOSCOPE is the closest system to BLAZEIT. NOSCOPE focuses on *binary detection*: the presence/absence of a given object class. Namely, NOSCOPE cannot directly answer queries in the form of counting or scrubbing for multiple instances of an object or objects.

As NOSCOPE is not directly applicable to the tasks we consider, where relevant, we compare against a NOSCOPE *oracle*, namely a method that returns (on a frame-by-frame basis) whether or not an object class is present in the scene. We assume the oracle is free to query. Thus, this oracle is strictly more powerful—both in terms of accuracy and speed—than NOSCOPE. We describe how the NOSCOPE oracle can be used to answer each type of query.

Aggregates. As NOSCOPE cannot distinguish between one and several objects, whenever NOSCOPE detects an object class is present, it must call the object detection method to identify the individual objects. Thus, to count cars in taipei would require performing object detection on 64.4% of the frames (i.e. the occupancy rate of cars).

Cardinality-limited scrubbing. As above, NOSCOPE can be used to filter frames that do not contain the objects of interest. For example, if the query were searching for at least one bus and at least five cars in taipei, NOSCOPE can be used to remove frames that do not have a bus and a car. Object detection will then be performed on the remaining frames until the requested number of events is found. Thus, for finding rare events, NOSCOPE fares poorly.

Content-based selection. NOSCOPE can only use label-based filtering, but not the other filters classes.

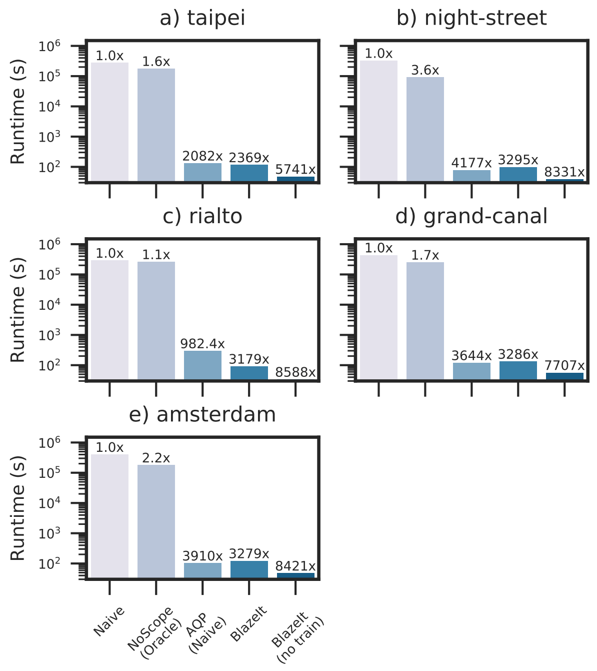

Figure 5: End-to-end runtime of baselines and BLAZEIT on aggregate queries where the query is rewritten with a specialized network, measured in seconds (log scale). All queries targeted an error of 0.1.

10.2 Aggregate Queries

We evaluate BLAZEIT on six aggregate queries across six videos. The queries are similar to Query 4a (shown in Section 2), with the video and object class changed. We ran five variants of each query:

- Naive: object detection on every frame.
- NOSCOPE oracle: the object detection method on every frame with the object class present.
- Naive AQP: sample from the video.
- BLAZEIT: we use specialized NNs and control variates for efficient sampling.
- BLAZEIT (no train): we exclude the training time from BLAZEIT.

There are two qualitatively different modes in which BLAZEIT executes these queries: 1) where BLAZEIT rewrites the query using a specialized NN, and 2) when BLAZEIT samples using specialized NNs as control variates, and select between these methods as described in Section 6. We analyze these cases separately.

Query rewriting via specialized NNs. We evaluate the runtime and accuracy of specialized networks when the query can be entirely rewritten by running the specialized NN instead. We ran each query with a target error rate of 0.1 and a confidence interval of 95%. We show the average of three runs. The results are shown in Figures 5. The specialized NNs were unable to achieve this accuracy target for archie, so

Video Name	Error
`taipei`	0.043
`night-street`	0.022
`rialto`	-0.031
`grand-canal`	0.081
`amsterdam`	0.050

Table 4: Average error of 3 runs of query-rewriting using a specialized NN for counting. These videos stayed within an error of 0.1.

Video Name	Pred (day 1)	Actual (day 1)	Pred (day 2)	Actual (day 2)
`taipei`	0.86	0.85	1.21	1.17
`night-street`	0.76	0.84	0.40	0.38
`rialto`	2.25	2.15	2.34	2.37
`grand-canal`	0.95	0.99	0.87	0.81

Table 5: Estimated and true counts for specialized NNs run on two different days of video.

we exclude it. However, we show below that specialized NNs can be used as a control variate even in this case.

As shown, BLAZEIT can achieve up to 8500× speedup if the model is cached and a 3200× speedup when including the training time and the time to compute thresholds. In contrast, [46] does not include this time in their evaluation. The NOSCOPE oracle baseline does not perform well when the video has many objects of interest (e.g., `rialto`).

In some cases, naive AQP outperform BLAZEIT when BLAZEIT trains the specialized NNs from scratch. However, in all cases, BLAZEIT outperform AQP when the models are cached.

While specialized NNs do not provide an error guarantee, we show that the absolute error stays within the 0.1 for the given videos in Table 4. Thus, we empirically demonstrate that specialized NNs can be used for query rewriting while respecting the user's error bounds.

Sampling and control variates. We evaluate the runtime and accuracy of using sampling and sampling with control variates. Because of the high computational cost of running object detection, we ran the object detection method once and recorded the results. Thus, the run times in this section are estimated from the number of object detection calls.

We targeted error rates of 0.01, 0.02, 0.03, 0.04, 0.05, and 0.1. Each query was run with a confidence interval of 95%. We average the number of samples for each error level over 100 runs.

The results are shown in Figure 6. As shown, using specialized NNs as a control variate can deliver up to a 2× reduction in sample complexity. As predicted by theory, the reduction in variance depends on the correlation coefficient between the specialized NNs and the object detection methods. Specifically, as the correlation coefficient increases, the sample complexity decreases.

Specialized NNs do not learn the average. A potential concern of specialized NNs is that they simply learn the average number of cars. To demonstrate that they do not, we swap the day of video for choosing thresholds and testing data. We show the true counts for each day and the average of 3 runs in Table 5. Notably, we see that the specialized NNs return different results for each day. This shows that the specialized NNs do not learn the average and return meaningful results.

10.3 Cardinality-limited Scrubbing Queries

We evaluate BLAZEIT on six scrubbing queries, in which frames of interest are returned to the user, up to the requested number of frames. The queries are similar to Query 4b, as shown in Section 2. We show in Table 6 query details and the number of instances of each query. If the user queries more than the maximum number of instances, BLAZEIT must query every frame. Thus, we chose queries with at least 10 instances.

In scrubbing queries, BLAZEIT will only return true positives (as it calls the full object detection method to verify frames of interest), thus

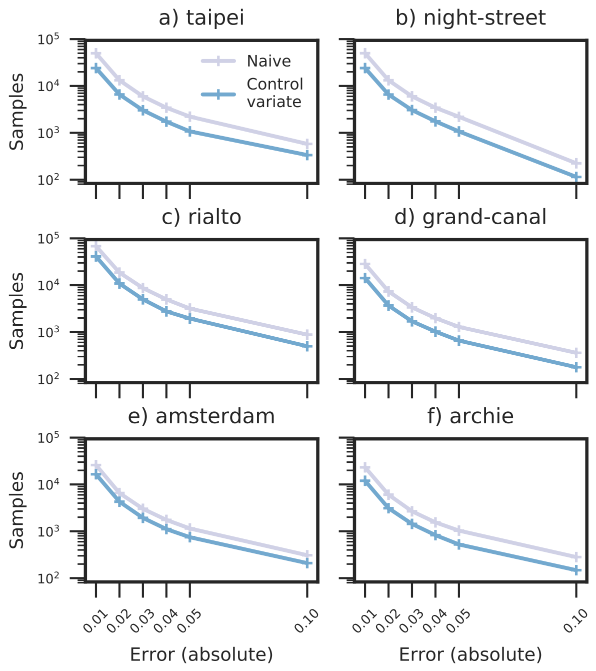

Figure 6: Sample complexity of naive AQP and AQP with control variates. Note the y-axis is on a log-scale.

Video name	Object	Number	Instances
`taipei`	car	6	70
`night-street`	car	5	29
`rialto`	boat	7	51
`grand-canal`	boat	5	23
`amsterdam`	car	4	86
`archie`	car	4	102

Table 6: Query details and number of instances. We selected rare events with at least 10 instances.

we only report the runtime. Additionally, if we suppose that the videos are pre-indexed with the output of the specialized NNs, we can simply query the frames using information from the index. This scenario might occur if, for example, the user executed an aggregate query as above. Thus, we additionally report sample complexity as an objective metric across object detection methods.

We run the following variants:

- Naive: the object detection method is run until the requested number of frames is found.
- NOSCOPE: the object detection method is run over the frames containing the object class(es) of interest until the requested number of frames is found.
- Sampling: the video is randomly sampled until the requested number of events is found.
- BLAZEIT: specialized NNs are used as a proxy signal to rank the frames (Section 7).
- BLAZEIT (indexed): we assume the specialized NN has been trained and run over the remaining data, as might happen if a user runs queries about some class repeatedly.

Single object class. As shown in Figure 7, BLAZEIT can achieve over a 1000× speedup compared to several baselines. We see that the non-specialized baselines do poorly in finding rare objects, where BLAZEIT's specialized NNs can serve as a high-fidelity signal.

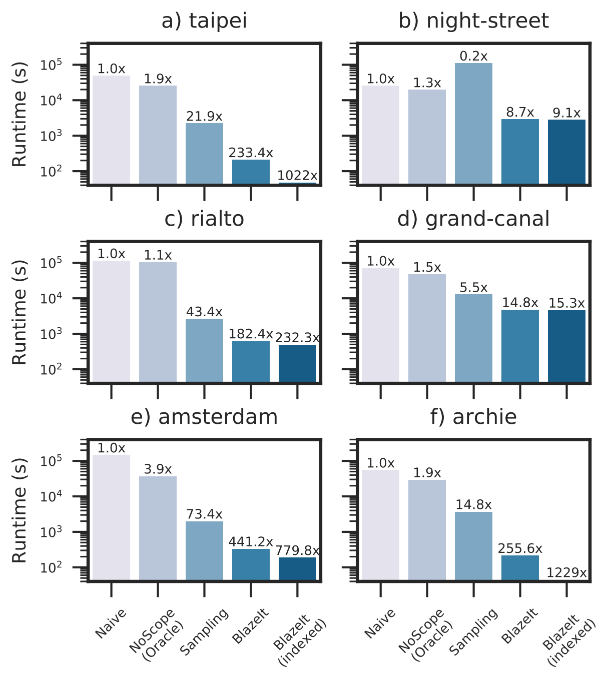

Figure 7: End-to-end runtime of baselines and BLAZEIT on scrubbing queries. Note the y-axis is on a log-scale. All queries looked for 10 events. The average over three runs is shown.

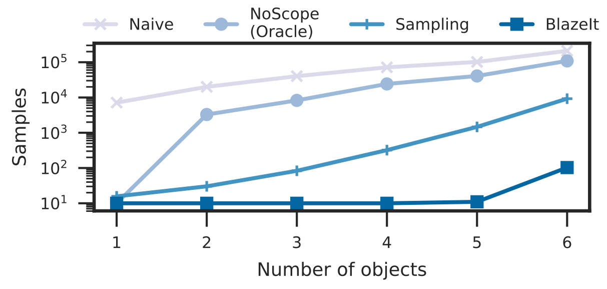

Figure 8: Sample complexity of baselines and BLAZEIT when searching for at least N cars in `taipei`. Note the y-axis is on a log-scale. All queries looked for 10 events.

We additionally vary the number of cars in `taipei` to see if BLAZEIT could also search for common objects. The results are shown in Figure 8. For both the naive method and the NOSCOPE oracle, the same complexity increases as the number of cars increases. However, for up to 5 cars, BLAZEIT's sample complexity remains nearly constant, which demonstrates the efficacy of biased sampling. While BLAZEIT shows degraded performance with 6 cars, there are only 70 such instances, and is thus significantly harder to find.

Multiple object classes. We additionally test BLAZEIT on multiple object classes by searching for at least one bus and at least five cars in `taipei`. There are 63 instances of such events in the test set.

The end-to-end speedups are shown in Figure 9. Searching for multiple object classes is favorable for the NOSCOPE oracle, as it becomes more selective. Nonetheless, BLAZEIT significantly outperforms the NOSCOPE oracle, giving up to a 81× performance increase. BLAZEIT also significantly outperforms the naive baseline, giving over a 966× speedup.

Additionally, we show the sample complexity as a function of the `LIMIT` in Figure 10 of BLAZEIT and the baselines, for `taipei`. We see that BLAZEIT can be up to 5 orders of magnitude more sample efficient over both the naive baseline and NoScope.

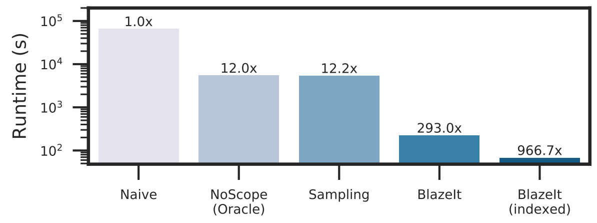

Figure 9: End-to-end runtime of baselines and BLAZEIT on finding at least one bus and at least five cars in `taipei`. Note the y-axis is on a log scale.

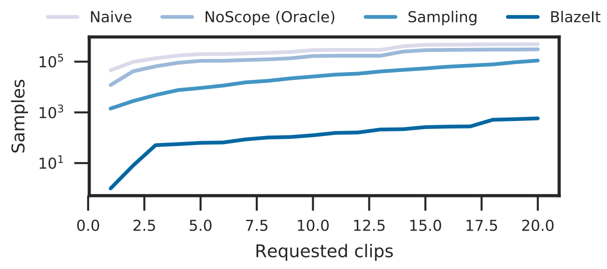

Figure 10: Sample complexity of BLAZEIT, NoScope and the naive method when searching for at least one bus and at least five cars in `taipei`. The x-axis is the number of requested frames. Note the y-axis is on a log scale.

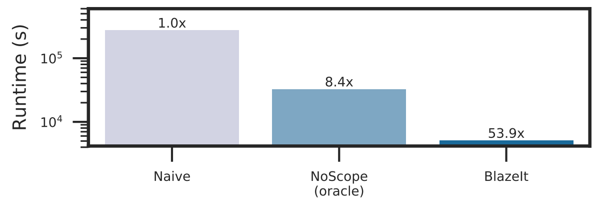

Figure 11: End-to-end throughput of baselines and BLAZEIT on the query in Figure 4c. Note the y-axis is on a log scale.

10.4 Content-based Selection Queries

To illustrate the effectiveness of content-based filters, we evaluate BLAZEIT on the query shown in Figure 4c.

We run the following variants:

- Naive: we run the object detection method on every frame.
- NOSCOPE oracle: we run the object detection method on the frames that contain the object class of interest.
- BLAZEIT: we apply the filters described in Section 8.

We do not include an AQP baseline as AQP is not applicable for exhaustive selection, as sampling does not help for exhaustive queries.

For each query, BLAZEIT's CBO trains, estimates the selectivity, and computes the threshold for each filter applicable to the query (which is determined by BLAZEIT's rule-based optimizer). We include the time to train the filters and select the thresholds in the runtime. Due to the large computational cost of running the object detector, we extrapolate its cost by multiplying the number of calls by the runtime of the object detector.

End-to-end performance. The results for the end-to-end runtime of the naive baseline, the NOSCOPE oracle, and BLAZEIT are shown in Figure 11. As buses are relatively rare (12% occupancy, see Table 3), NOSCOPE performs well on this query, giving a 8.4× performance

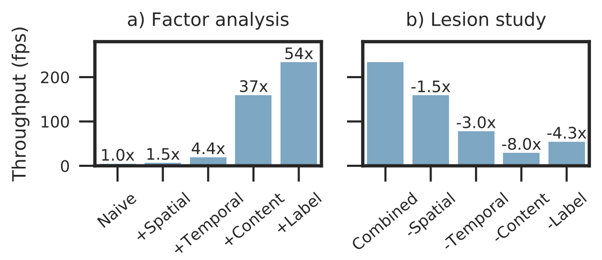

Figure 12: Factor analysis and lesion study of BLAZEIT's filters on the query in Figure 4c. In the factor analysis, we add filters one at a time. In the lesion study, the filters are individually removed. The filters are described in Section 8.

improvement over the naive method. However, BLAZEIT outperforms the NOSCOPE oracle by 6.4×, due to its extended classes of filters. Furthermore, BLAZEIT delivers up to 54× improved throughput over naive methods for this query.

Factor analysis. We perform a factor analysis and lesion study to understand the impact of each class of filter. In the factor analysis, we added the filters one at a time. In the lesion study, we individually removed the filters. Results are shown in Figure 12. As shown in the factor analysis, every filter adds a non-trivial speedup. Additionally, removing any class of filter reduces performance. Thus, every class of filter improves performance for this query.

11 Related Work

BLAZEIT builds on a long tradition of data management for multimedia and video, and on recent advances in computer vision. We outline some relevant parts of the literature below.

Approximate Query Processing. In AQP systems, the result of a query is returned significantly faster by subsampling the data [22]. Typically, the user specifies an error bound [5], or the error bound is refined over time [37]. Prior work has leveraged various sampling methods [4, 11], histograms [3, 15, 29, 63], and sketches [10, 38, 43].

Other contemporary work use filters with a false negative rate (called probabilistic predicates) that are automatically learned from a hold-out set [58]. These could be incorporated into BLAZEIT, but in this work, we focus on filters that can be automatically extracted from the query, as opposed to arbitrary predicates.

We leverage ideas from this space and introduce a new form of variance reduction in the form of control variates [30] by using specialized networks. This form of variance reduction, and others involving auxiliary variables, does not make sense in a traditional relational database: the cost of materializing a tuple must be disproportionally large compared to computing the auxiliary variable.

Additionally, we use specialized NNs as a form of importance sampling to bias the search for cardinality-limited scrubbing queries.

Visual data management. Visual data management has aimed to organize and query visual data, starting from systems such as Chabot [61] and QBIC [21]. These systems were followed by a range of "multimedia" database for storing [8, 52], querying [7, 50, 62], and managing [26, 42, 73] video data. Many of these systems use classic computer vision techniques such as *low-level* image features (e.g colors, textures) and rely on textual annotations for semantic queries. However, recent advances in computer vision allow the *automatic* population of semantic data and thus we believe it is critical to reinvestigate these systems.

Visual query languages. Many query languages for visual data have been developed [17, 41, 57]. However, much of the prior work has assumed the data is provided (e.g., actors in a movie) or that low-level image features are queried. In this work, we explicitly choose to use standard SQL with simple extensions in FRAMEQL and focus on how these fields can be *automatically populated* rather than on the syntax of the language.

Modern video analytics. Systems builders have created systems for analyzing video; perhaps the most related is NOSCOPE [46]. NOSCOPE is a highly tuned pipeline for binary detection: it returns the presence or absence of a particular object class in video. Similar to NOSCOPE, other systems, such as FOCUS [40] and TAHOMA [6] have optimized binary detection. However, these systems are inflexible and cannot adapt to user's queries. Additionally, as NOSCOPE does not focus on the exploratory setting, it does not aim to minimize the training time of specialized NNs. In BLAZEIT, we leverage and extend specialization and present novel optimizations for aggregation, scrubbing, and content-based selection, which these systems do not support.

Other systems, such as VideoStorm [74], aim to reduce latency of live queries that are pre-defined *as a computation graph*. As the computation is specified as a black-box, VideoStorm does not have access to the semantics of the computation to perform certain operations, such as in BLAZEIT. In BLAZEIT, we introduce FRAMEQL and an optimizer that can infer optimizations from the given query. Additionally, we focus on the batch analytics setting. However, we believe BLAZEIT could be run on a system such as VideoStorm for live analytics.

In the batch setting, SCANNER [65] takes a pre-defined computation graph and executes the graph using all the hardware resources available. However, SCANNER also does not have access to the semantics of the computation and thus cannot perform certain types of optimizations or use specialization. However, we believe BLAZEIT could be run on SCANNER for scale-out.

Speeding up deep networks. We briefly discuss two of the many forms of improving deep network efficiency.

First, a large body of work changes model architecture or weights for improved inference efficiency, that preserve the full generality of these models. Model compression uses a variety of techniques from pruning [32] to compressing [12] weights from the original model, which can be amenable to hardware acceleration [31]. Model distillation uses a large model to train a smaller model [39]. However, these methods aim to retain or nearly retain the accuracy of the original model. These methods do not allow for adapting to the task at hand, as BLAZEIT does. Additionally, these methods are largely orthogonal to BLAZEIT, and reducing the cost of object detection would also improve BLAZEIT's runtime.

Second, model specialization [46, 70] aims to improve inference speeds by training a small model to mimic a larger model *on a reduced task*. However, specialization has typically been applied in *specific* pipelines, such as NOSCOPE's binary detection. In BLAZEIT, we leverage and extend specialization to counting and multi-class classification.

12 Conclusions

Querying video for semantic information has become possible with recent advances in computer vision, but these models run as much as 10× slower than real-time. Additionally, deploying these models requires complex programming with low-level libraries. In response, we present a declarative language for video analytics, FRAMEQL, and BLAZEIT, a system that accepts, automatically optimizes, and executes FRAMEQL queries up to three orders of magnitude faster. We demonstrate that FRAMEQL can answer a range of real-world queries, of which we focus on exploratory queries in the form of aggregates and searching for rare events. BLAZEIT introduces new techniques based on AQP, Monte Carlo sampling, and rare-event simulation, and extends specialization to answer these exploratory queries up to three orders of magnitude faster. These results suggest that large video datasets can be explored with orders of magnitude lower computational cost.

13 References

[1] Cornell lab bird cams. `http://cams.allaboutbirds.org/`.

[2] Cctv: Too many cameras useless, warns surveillance watchdog tony porter, 2015.

[3] S. Acharya, P. B. Gibbons, and V. Poosala. Aqua: A fast decision support systems using approximate query answers. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 754–757. Morgan Kaufmann Publishers Inc., 1999.

[4] S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. Jordan, S. Madden, B. Mozafari, and I. Stoica. Knowing when you're wrong: building fast and reliable approximate query processing systems. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 481–492. ACM, 2014.

[5] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 29–42. ACM, 2013.

[6] M. R. Anderson, M. Cafarella, T. F. Wenisch, and G. Ros. Predicate optimization for a visual analytics database. *SysML*, 2018.

[7] W. Aref, M. Hammad, A. C. Catlin, I. Ilyas, T. Ghanem, A. Elmagarmid, and M. Marzouk. Video query processing in the vdbms testbed for video database research. In *Proceedings of the 1st ACM international workshop on Multimedia databases*, pages 25–32. ACM, 2003.

[8] F. Arman, A. Hsu, and M.-Y. Chiu. Image processing on compressed data for large video databases. In *Proceedings of the first ACM international conference on Multimedia*, pages 267–272. ACM, 1993.

[9] J. A. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava. Participatory sensing. In *WSW*, 2006.

[10] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

[11] S. Chaudhuri, G. Das, and V. Narasayya. Optimized stratified sampling for approximate query processing. *ACM Transactions on Database Systems (TODS)*, 32(2):9, 2007.

[12] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015.

[13] C. Coleman, D. Kang, D. Narayanan, L. Nardi, T. Zhao, J. Zhang, P. Bailis, K. Olukotun, C. Re, and M. Zaharia. Analysis of dawnbench, a time-to-accuracy machine learning performance benchmark. *arXiv preprint arXiv:1806.01427*, 2018.

[14] C. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia. Dawnbench: An end-to-end deep learning benchmark and competition. *Training*, 100(101):102, 2017.

[15] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Space-and time-efficient deterministic algorithms for biased quantiles over data streams. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 263–272. ACM, 2006.

[16] J. De Cea and E. Fernández. Transit assignment for congested public transport systems: an equilibrium model. *Transportation science*, 27(2):133–147, 1993.

[17] U. Demir, M. Koyuncu, A. Yazici, T. Yilmaz, and M. Sert. Flexible content extraction and querying for videos. In *International Conference on Flexible Query Answering Systems*, pages 460–471. Springer, 2011.

[18] M. E. Dönderler, E. Şaykol, U. Arslan, Ö. Ulusoy, and U. Güdükbay. Bilvideo: Design and implementation of a video database management system. *Multimedia Tools and Applications*, 27(1):79–104, 2005.

[19] B. Efron and R. J. Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

[20] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2010.

[21] 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.

[22] M. N. Garofalakis and P. B. Gibbons. Approximate query processing: Taming the terabytes. In *VLDB*, 2001.

[23] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[24] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE, 2012.

[25] S. Geisser. *Predictive inference*. Routledge, 2017.

[26] S. Gibbs, C. Breiteneder, and D. Tsichritzis. Audio/video databases: An object-oriented approach. In *Data Engineering, 1993. Proceedings. Ninth International Conference on*, pages 381–390. IEEE, 1993.

[27] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[28] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He. Detectron. `https://github.com/facebookresearch/detectron`, 2018.

[29] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *ACM SIGMOD Record*, volume 30, pages 58–66. ACM, 2001.

[30] J. M. Hammersley and D. C. Handscomb. General principles of the monte carlo method. In *Monte Carlo Methods*, pages 50–75. Springer, 1964.

[31] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 243–254. IEEE, 2016.

[32] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[33] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 123–136. ACM, 2016.

[34] D. Haussler and E. Welzl. -nets and simplex range queries. *Discrete & Computational Geometry*, 2(2):127–151, 1987.

[35] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.

[36] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning

for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[37] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *Acm Sigmod Record*, volume 26, pages 171–182. ACM, 1997.

[38] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. *External memory algorithms*, 50:107–118, 1998.

[39] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[40] K. Hsieh, G. Ananthanarayanan, P. Bodik, P. Bahl, M. Philipose, P. B. Gibbons, and O. Mutlu. Focus: Querying large video datasets with low latency and low cost. *arXiv preprint arXiv:1801.03493*, 2018.

[41] E. Hwang and V. Subrahmanian. Querying video libraries. *journal of visual communication and image representation*, 7(1):44–60, 1996.

[42] R. Jain and A. Hampapur. Metadata in video databases. *ACM Sigmod Record*, 23(4):27–33, 1994.

[43] C. Jin, W. Qian, C. Sha, J. X. Yu, and A. Zhou. Dynamically maintaining frequent items over a data stream. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 287–294. ACM, 2003.

[44] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

[45] S. Juneja and P. Shahabuddin. Rare-event simulation techniques: an introduction and recent advances. *Handbooks in operations research and management science*, 13:291–350, 2006.

[46] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia. Noscope: optimizing neural network queries over video at scale. *Proceedings of the VLDB Endowment*, 10(11):1586–1597, 2017.

[47] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, pages 1725–1732, 2014.

[48] T. C. Kuo and A. L. Chen. A content-based query language for video databases. In *Multimedia Computing and Systems, 1996., Proceedings of the Third IEEE International Conference on*, pages 209–214. IEEE, 1996.

[49] T. C. Kuo and A. L. Chen. Content-based query processing for video databases. *IEEE Transactions on Multimedia*, 2(1):1–13, 2000.

[50] M. La Cascia and E. Ardizzone. Jacob: Just a content-based query system for video databases. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 2, pages 1216–1219. IEEE, 1996.

[51] T.-L. Le, M. Thonnat, A. Boucher, and F. Brémond. A query language combining object features and semantic events for surveillance video retrieval. In *International Conference on Multimedia Modeling*, pages 307–317. Springer, 2008.

[52] J. Lee, J. Oh, and S. Hwang. Strg-index: Spatio-temporal region graph indexing for large video databases. In *SIGMOD*, pages 718–729. ACM, 2005.

[53] T. Lee. Theres growing evidence teslas autopilot handles lane dividers poorly, 2018.

[54] J. Z. Li, M. T. Ozsu, D. Szafron, and V. Oria. Moql: A multimedia object query language. In *Proceedings of the 3rd International Workshop on Multimedia Information Systems*, pages 19–28, 1997.

[55] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[56] S. Lohr. *Sampling: design and analysis*. Nelson Education, 2009.

[57] C. Lu, M. Liu, and Z. Wu. Svql: A sql extended query language for video databases. *International Journal of Database Theory and Application*, 8(3):235–248, 2015.

[58] Y. Lu, A. Chowdhery, S. Kandula, and S. Chaudhuri. Accelerating machine learning inference with probabilistic predicates. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1493–1508. ACM, 2018.

[59] J. Matejka, T. Grossman, and G. Fitzmaurice. Swift: reducing the effects of latency in online video scrubbing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 637–646. ACM, 2012.

[60] J. Matejka, T. Grossman, and G. Fitzmaurice. Swifter: improved online video scrubbing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1159–1168. ACM, 2013.

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

[62] J. Oh and K. A. Hua. Efficient and cost-effective techniques for browsing and indexing large video databases. In *ACM SIGMOD Record*, volume 29, pages 415–426. ACM, 2000.

[63] G. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. *ACM Sigmod Record*, 14(2):256–276, 1984.

[64] H. Pirahesh, J. M. Hellerstein, and W. Hasan. Extensible/rule based query rewrite optimization in starburst. In *ACM Sigmod Record*, volume 21, pages 39–48. ACM, 1992.

[65] A. Poms, W. Crichton, P. Hanrahan, and K. Fatahalian. Scanner: Efficient video analysis at scale (to appear). 2018.

[66] J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger. *arXiv preprint*, 2017.

[67] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[68] C. P. Robert. *Monte carlo methods*. Wiley Online Library, 2004.

[69] A. W. Senior, L. Brown, A. Hampapur, C.-F. Shu, Y. Zhai, R. S. Feris, Y.-L. Tian, S. Borger, and C. Carlson. Video analytics for retail. In *Advanced Video and Signal Based Surveillance, 2007. AVSS 2007. IEEE Conference on*, pages 423–428. IEEE, 2007.

[70] H. Shen, S. Han, M. Philipose, and A. Krishnamurthy. Fast video classification via adaptive cascading of deep models. *arXiv preprint*, 2016.

[71] X. Sun, L. Muñoz, and R. Horowitz. Highway traffic state estimation using improved mixture kalman filters for effective ramp metering control. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, volume 6, pages 6333–6338. IEEE, 2003.

[72] L. Wasserman. *All of statistics: a concise course in statistical inference*. Springer Science & Business Media, 2013.

[73] A. Yoshitaka and T. Ichikawa. A survey on content-based retrieval for multimedia databases. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):81–93, 1999.

[74] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman. Live video analytics at scale with approximation and delay-tolerance. In *NSDI*, volume 9, page 1, 2017.

[75] X. Zhu, Y. Wang, J. Dai, L. Yuan, and Y. Wei. Flow-guided feature aggregation for video object detection. *arXiv preprint arXiv:1703.10025*, 2017.

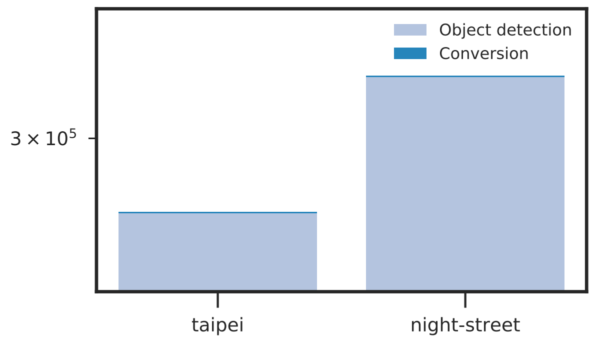

Figure 13: Time of object detection and conversion to FRAMEQL records, when processing the entire video. Zoomed in and log-scaled for visibility.

APPENDIX

A FRAMEQL

A.1 Populating FRAMEQL

In this work, we focus on the batch analytics setting (as opposed to streaming analytics) so we assume the all the video is available for processing. To our knowledge, the majority of object detection and entity resolution methods are deterministic. Thus, once the object detection and entity resolution methods are fixed, the FRAMEQL records do not change.

We have empirically found that, even for busy scenes, a fully populated FRAMEQL table has around 100k rows per hour of video. We show the breakdown of time for detection and processing the data to FRAMEQL records for `taipei` and `night-street` in Figure 13 (`taipei` is the busiest video and thus takes the longest time to convert the output of object detection to FRAMEQL records). As processing the output of object detection to FRAMEQL records is a tiny fraction of total time, we ignore the transformation time.

A.2 Other Visual Query Languages

There are many other visual query languages in the literature, including SVQL [57], CVQL [49], BVQL [18], and SRQL [51]. FRAMEQL shares similarities with many of these query languages, but has several key differences. First, the majority of prior visual query languages either used low-level features, assumed the semantic data was provided, or used classical computer vision methods to extract semantic data. Second, many of these query languages focuses on events or specific instances of objects (e.g., specific actors in a movie). Third, many prior query languages implicitly focus on relatively short videos (e.g., minutes) as opposed to the long-running video streams we consider in this work.

In designing FRAMEQL, we specifically aimed to avoid the user having to annotate video. Thus, we restricted the language to only allow expressions that can be *automatically* computed with high accuracy (e.g., we do not allow querying for specific events). Techniques to extract this data automatically from video with high accuracy have only been developed recently [35].

A.3 FRAMEQL examples

We give further examples of FRAMEQL.

First, counting the number of distinct cars can be written as:

```
SELECT COUNT (DISTINCT trackid)
FROM taipei
WHERE class = 'car'
```

which is not the same as counting the average number of cars in a frame, as this query looks for distinct instances of cars using `trackid` (cf. Figure 4a).

Second, error rates can be set using syntax similar to BlinkDB [5]:

```
SELECT COUNT(*)
FROM taipei
WHERE class = 'car'
ERROR WITHIN 0.1 CONFIDENCE 95%
```

Third, NOSCOPE can be replicated as FRAMEQL queries of the form:

```
SELECT timestamp
FROM taipei
WHERE class = 'car'
FNR WITHIN 0.01
FPR WITHIN 0.01
```

Finally, a UDF could be used to classify cars into subtypes:

```
SELECT *
FROM taipei
WHERE class = 'car'
  AND classify(content) = 'sedan'
```

A.4 Extensions to FRAMEQL

In this work, we focus on queries that can be optimized using current techniques. However, many use cases require two features that we do not currently support: 1) joins and 2) global identifiers for objects. As FRAMEQL is based on standard SQL and relational algebra, these can easily be added to the language. Thus, we describe how to add joins and global identifiers to FRAMEQL.

Joins. Joins can be added to FRAMEQL as in standard SQL. Joins can be used to answer a wider range of queries. For example, computing the average count of cars on a single street camera can be used to determine when the given street is busiest, but not when traffic is the highest across a city. Joining on the timestamp and computing the average number of cars could answer this query.

Global identification. While FRAMEQL can express a wide range of queries, several types of queries require a unique global identifier. For example, if two cameras were placed a mile apart on a highway, the flow rate of cars on the highway could be computed by the average time it takes for a car to leave one camera and arrive at the other.

Thus, FRAMEQL could be extended by adding a field `globalid`. In the case of cars, the license plate number could be used as a global identifier, but computing `globalid` is a difficult task in general.

B Aggregation

B.1 Proof of Correctness

Correctness of AQP. As the correctness of AQP has been extensively studied, we sketch the proof outline here, which has been slightly modified from [37].

As BLAZEIT does not currently support joins, all sampling strategies draw iid from the frames (or boxes). Furthermore, we assume the quantities of interest are bounded, which is trivially true, except in the case where a UDF returns an invalid value or infinity. We ignore this case.

Denote the random variable over n samples as Y_n. As the samples are drawn iid and are bounded, the central limit theorem applies and thus Y_n converges to the correct quantity [72].

Correctness of Control Variates. The proof of correctness of control variates follows directly from the above proof, as control variates are an unbiased estimator of the same quantity.

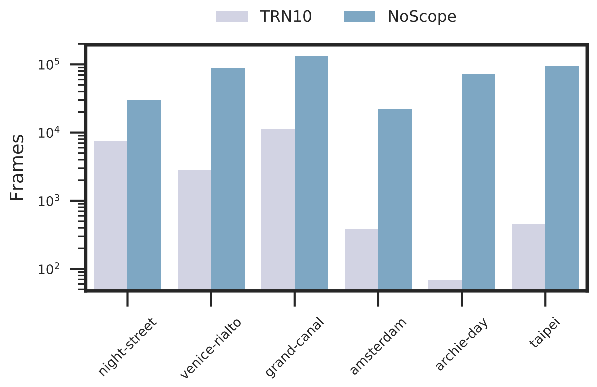

Figure 15: Number of samples to find 10 of the requested objects for each query, using TRN10 or a representative NoScope model. As shown, TRN10 significantly outperforms the NoScope model on all videos. The y-axis is on a log-scale. Average of 3 runs.

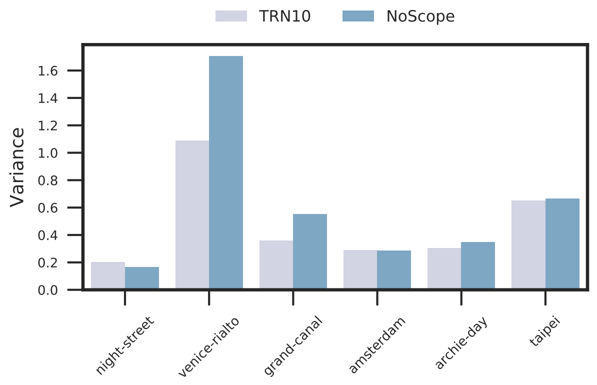

Figure 16: Variance of the control variates estimator when using TRN10 or a representative NoScope model (lower is better). As shown, TRN10 typically matches or beats the NoScope model, except for `night-street`.

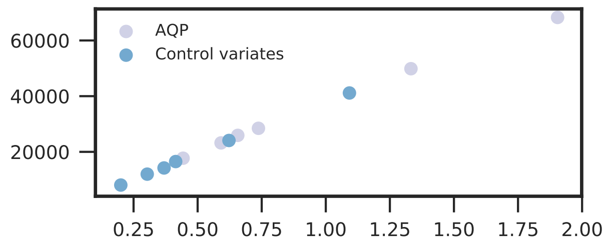

Figure 14: Variance of the estimator vs number of samples.

B.2 Extensions to Aggregation

In this work, we focus on extending aggregation beyond existing techniques, but we describe how aggregation can speed up other statistics.

Aggregation for occupancy. Another statistic of interest is the percentage of frames that are occupied, which can be written in FrameQL as:

```
SELECT FCOUNT DISTINCT(*)
FROM taipei
WHERE class = 'car'
ERROR WITHIN 0.01 CONFIDENCE 95%
```

To answer this query, BlazeIt can perform Algorithm 1, but instead train a specialized NN that performs binary detection instead of counting the number of objects in the frame.

Aggregation for number of unique objects. Another statistic of interest is the number of unique objects, which can be written in FrameQL as:

```
SELECT COUNT (DISTINCT trackid)
FROM taipei
WHERE class = 'car'
```

To answer this query, BlazeIt can use standard AQP with the following sampling procedure:

- Sample i iid from the number of frames
- For each sampled frame f_i, perform object detection and entity resolution on frame f_i and f_{i+1}
- Return differential of the number of objects that are in frame f_i but not in f_{i+i}

Aggregation for box statistics. Another class of statistics are statistics over the bounding boxes. For example, an analyst might be interested in the average box area or the average center coordinate. For any numerical statistic over the bounding box, BlazeIt can leverage traditional AQP, using the following procedure:

- Sample a frame f_i iid.
- Perform object detection of f_i
- Sample iid a single box matching the predicates of the query
- Run the UDF over the box, return the answer

To increase efficiency, BlazeIt can cache the box information if the frame f_i is sampled again.

C Further Experiments

Variance of estimators vs number of samples. Theoretical analysis predicts that the variance of an estimator (either random sampling or control variates) linearly affects the number of samples to achieve a given error rate. To verify this hypothesis, we show a plot of the variance of the estimator vs the number of samples necessary to achieve an error of 0.01 at a confidence interval of 95%. As shown in Figure 14, the number of samples is linear with the variance, as predicted by theory. We can also see that control variates has lower variance than traditional AQP, as predicted by theory.

Effect of NN type. In this work, we chose to use a tiny ResNet (referred to as TRN10) as the default specialized architecture. ResNets are an extremely popular architecture [13, 14]. To test our hypothesis that TRN10 is a good default, we compared TRN10 to a representative NoScope model, parameterized by 32 base filters, 32 dense neurons, and 4 layers.

We used TRN10 and the NoScope model on scrubbing tasks for each of the videos and computed the number of samples required to find the requested number of events in Table 6. As shown in Figure 15, TRN10 requires significantly fewer samples compared to the NoScope model on all videos.

We additionally used TRN10 and the NoScope for the aggregation tasks for each video and computed the variance of the control variate estimator (the variance of the estimator is directly related to the number of samples; lower is better). As shown in Figure 16, TRN10 typically matches or beats the NOSCOPE model, except for `night-street`.

While these results suggest that TRN10 is a good default for BLAZEIT, we have not fully explored model search for exploratory visual queries. We view this as an exciting area of future research.

D Rule-based Optimizer

BLAZEIT currently uses a rule-based optimizer for queries, as we have found it sufficient to optimize a large class of queries. BLAZEIT's rule-based optimizer attempts to classify queries as (approximate) aggregation, cardinality-limited scrubbing, or content-based selection and applies the rules described in the main paper in these settings. For all other queries, BLAZEIT falls back to materializing all the rows in the FRAMEQL table.

To determine if a query is an approximate aggregation query (exact aggregation requires all the relevant rows to be materialized), BLAZEIT inspects the FRAMEQL query for the `ERROR` keyword and an aggregation keyword (e.g., `FCOUNT` or `AVG`). BLAZEIT will then determine the necessary fields and perform approximate aggregation as described above.

To determine if a query is a cardinality-limited scrubbing query, BLAZEIT inspects the FRAMEQL query for the `LIMIT` keyword. BLAZEIT will then determine the necessary fields for the query and perform cardinality-limited scrubbing as described above.

To determine if the query is a content-based selection (with applicable filters), BLAZEIT will inspect the query for the predicates as described in Section 8 and apply them as described above.

In all other cases, BLAZEIT will default to applying object detection to every frame.