BlazeIt: Optimizing Declarative Aggregation and Limit Queries for Neural Network-Based Video Analytics

Daniel Kang, Peter Bailis, Matei Zaharia
Stanford DAWN Project
blazeit@cs.stanford.edu

ABSTRACT

Recent advances in neural networks (NNs) have enabled automatic querying of large volumes of video data with high accuracy. While these deep NNs can produce accurate annotations of an object's position and type in video, they are computationally expensive and require complex, imperative deployment code to answer queries. Prior work uses approximate filtering to reduce the cost of video analytics, but does not handle two important classes of queries, aggregation and limit queries; moreover, these approaches still require complex code to deploy. To address the computational and usability challenges of querying video at scale, we introduce BLAZEIT, a system that optimizes queries of spatiotemporal information of objects in video. BLAZEIT accepts queries via FRAMEQL, a declarative extension of SQL for video analytics that enables video-specific query optimization. We introduce two new query optimization techniques in BLAZEIT that are not supported by prior work. First, we develop methods of using NNs as control variates to quickly answer approximate aggregation queries with error bounds. Second, we present a novel search algorithm for cardinality-limited video queries. Through these these optimizations, BLAZEIT can deliver up to 83× speedups over the recent literature on video processing.

PVLDB Reference Format:
Daniel Kang, Peter Bailis, Matei Zaharia. BlazeIt: Optimizing Declarative Aggregation and Limit Queries for Neural Network-Based Video Analytics. *PVLDB*, 13(4): 533 - 546, 2019.
DOI: https://doi.org/10.14778/3372716.3372725

1. INTRODUCTION

Two trends have caused recent interest in video analytics. First, cameras are now cheap and widely deployed, e.g., London alone has over 500,000 CCTVs [2]. Second, deep neural networks (DNNs) can automatically produce annotations of video. For example, object detection DNNs [20] return a set of bounding boxes and object classes given an image or frame of video. Analysts can use these DNNs to extract object positions and types from every frame of video, a common analysis technique [62]. In this work, we study the batch setting, in which large quantities of video are collected for later analysis [6, 37, 46].

While DNNs are accurate [32], naively employing them has two key challenges. First, from a usability perspective, these methods require complex, imperative programming across many low-level libraries, such as OpenCV, Caffe2, and Detectron [26]—an ad-hoc, tedious process. Second, from a computational perspective, the naive method of performing object detection on every frame of video is cost prohibitive at scale: state-of-the-art object detection, e.g., Mask R-CNN [32], runs at 3 frames per second (fps), which would take 8 GPU-decades to process 100 camera-months of video.

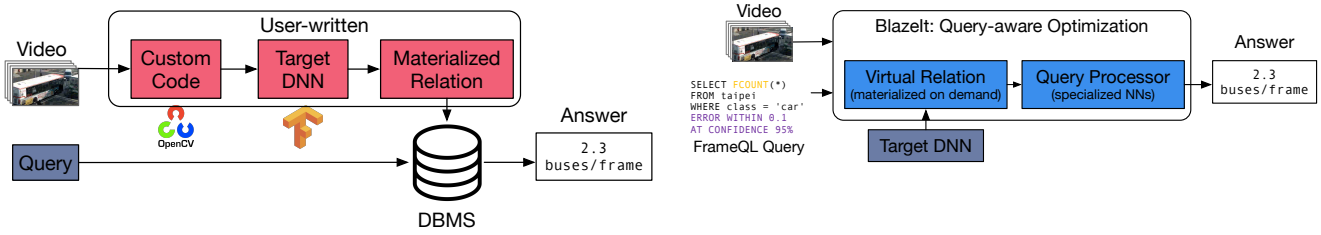
Researchers have recently proposed optimizations for video analytics, largely focusing on filtering via approximate predicates [6,9, 46,55]. For example, NOSCOPE and TAHOMA train cheaper, proxy models for filtering [6, 46]. However, these optimizations do not handle two key classes of queries: aggregate and limit queries. For example, an analyst may want to count the average number of cars per frame (aggregate query) or manually inspect only 10 instances of a bus and five cars (limit query) to understand congestion. Approximate filtering is inefficient for these queries, e.g., filtering for cars will not significantly speed up counting cars if 90% of the frames contain cars. Furthermore, these optimizations still require non-expert users to write complex code to deploy.

To address these usability and computational challenges, we present BLAZEIT, a video analytics system with a declarative query language and two novel optimizations for aggregation and limit queries. BLAZEIT's declarative query language, FRAMEQL, extends SQL with video-specific functionality and allows users familiar with SQL to issue video analytics queries. Since queries are expressed declaratively, BLAZEIT can *automatically* optimize them end-to-end with its query optimizer and execution engine. Finally, BLAZEIT provides two novel optimizations for aggregation and limit queries that outperforms prior work, including NOSCOPE [46] and approximate query processing (AQP), by up to 83×.

FRAMEQL allows users to query information of objects in video through a *virtual* relation. Instead of fully materializing the FRAMEQL relation, BLAZEIT uses optimizations to reduce the number of object detection invocations while meeting an accuracy guarantee based on the specification of the FRAMEQL query (Figure 1). FRAMEQL's relation represents the information of positions and classes of objects in the video. Given this relation, FRAMEQL can express selection queries in prior work [6,9,46,55], along with new classes of queries, including aggregation and limit queries (§4).

Our first optimization, to answer aggregation queries, uses query-specific NNs (i.e., specialized NNs [46]) as a control variate or to directly answer queries (§6). Control variates are a variance reduction technique that uses an auxiliary random variable that is corre-

Proceedings of the VLDB Endowment, Vol. 13, No. 4
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3372716.3372725



(a) Schematic of the naive method of querying video. Naively using DNNs or human annotators is too expensive for many applications.

(b) Schematic of BLAZEIT. BLAZEIT will create optimized query plans and avoid calling the expensive DNN where possible.

Figure 1: Schematic of the naive method of querying video and BLAZEIT. BLAZEIT does not require writing complex code and does not require pre-materializing all the tuples.

lated with the statistic of interest to reduce the number of samples necessary for a given error bound [28]. We show how to use specialized NNs as a control variate, a novel use of specialized NNs (which have been used for approximate filtering). In contrast, standard random sampling does not leverage proxy models and prior work (filtering) is inefficient when objects appear frequently.

Our second optimization, to answer cardinality-limited queries (e.g., a `LIMIT` query searching for 10 frames with at least three cars), evaluates object detection on frames that are more likely to contain the event using proxy models (§7). By prioritizing frames to search over, BLAZEIT can achieve exact answers while speeding up query execution. In contrast, filtering is inefficient for frequent objects and random sampling is inefficient for rare events.

Importantly, both of our optimizations provide exact answers or accuracy guarantees *regardless of the accuracy of the specialized NNs*. Furthermore, both of these optimizations can be extended to account for query predicates.

BLAZEIT incorporates these optimizations in an end-to-end system with a rule-based query optimizer and execution engine that efficiently executes FRAMEQL queries. Given query contents, BLAZEIT will generate an optimized query plan that avoids executing object detection wherever possible, while maintaining the user-specified accuracy (relative to the object detection method as ground truth).

We evaluate BLAZEIT on a variety of queries on four video streams that are widely used in studying video analytics [9, 37, 40, 46, 70] and two new video streams. We show that BLAZEIT can achieve up to 14× and 83× improvement over prior work in video analytics and AQP for aggregation and limit queries respectively.

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, including aggregation, limit, and selection queries.
2. We introduce an aggregation algorithm that uses control variates to leverage specialized NNs for more efficient aggregation than existing AQP methods by up to 14×.
3. We introduce an algorithm for limit queries that uses specialized NNs and can deliver up to 83× speedups over recent work in video analytics and random sampling.

2. EXAMPLE USE CASES

Recall that we focus on the batch setting in this work. We give several scenarios where BLAZEIT could be applied:

Urban planning. Given a set of traffic cameras at street corners, an urban planner performs traffic metering based on the number of cars, and determines the busiest times [69]. The planner is interested in how public transit interacts with congestion [16] and looks for 10 events 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 (see Figure 3).

Autonomous vehicle analysis. An analyst studying AVs notices anomalous behavior when the AV is in front of a yellow light and there are multiple pedestrians in the crosswalk [23], and searches for such events.

Store planning. A retail store owner places a CCTV in the store [66]. The owner segments the video into aisles and counts the number of people that walk through each aisle to understand the flow of customers. This information can be used for planning store and aisle layout.

```
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 300 frames apart (10s at 30 fps).

```
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 (15 frames). The last constraint is for noise reduction.

Figure 2: Three FRAMEQL example queries. As shown, the syntax is largely standard SQL.

(a) Red tour bus.

(b) White transit bus.

Figure 3: Examples of buses in `taipei`. A city planner might be interested in distinguishing tour buses from transit buses and uses color as a proxy.

Ornithology. An ornithologist (a scientist who studies birds) is interested in understanding bird feeding patterns, so places a webcam in front of a bird feeder [1]. Then, the ornithologist puts different bird feed on the left and right side of the feeder and counts the number of birds that visit each side. Finally, 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 user-defined functions (UDFs) over the content of the boxes. Thus, these applications illustrate a need for a unified method of expressing such queries.

3. BLAZEIT SYSTEM OVERVIEW

BLAZEIT's goal is to execute FRAMEQL queries as quickly as possible; we describe FRAMEQL in §4. To execute FRAMEQL queries, BLAZEIT uses a *target object detection method*, an entity resolution method, and the optional user-defined functions (UDFs). We describe the specification of these methods in this section and describe our defaults in §8. Importantly, we assume the object detection class types are provided.

BLAZEIT executes queries quickly by avoiding materialization using the techniques described §6 and §7. BLAZEIT uses proxy models, typically specialized neural networks [46,68], to avoid materialization (Figure 1b), which we describe below.

3.1 Components

Configuration. We assume the target object detection method is implemented with the following API:

```
OD(frame) → Set<Tuple<class, box>>    (1)
```

and the object classes (i.e., types) are provided. We assume the entity resolution takes two nearby frames and boxes and returns true if the boxes correspond to the same object. While we provide defaults (Section 8), the object detection and entity resolution methods 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. For example, to compute the "redness" of an object, a UDF could average the red channel of the pixels.

Target-model annotated set (TMAS). At ingestion time, BLAZEIT will perform object detection over a small sample of frames of the video with the target object detection NN and will store the metadata as FRAMEQL tuples, which we call the *target-model annotated set (TMAS)*. This procedure is done when the data is ingested and not per query, namely it is performed once, offline, and shared for multiple queries later. For a given query, BLAZEIT will use this metadata to materialize training data to train a query-specific proxy model; details are given in §6 and §7. The TMAS is split into training data and held-out data.

Proxy models and specialized NNs. BLAZEIT can infer proxy models and/or filters from query predicates, many of which must be trained from data. These proxy models can be used to accelerate query execution *with accuracy guarantees*.

Throughout, we use specialized NNs [46,67], specifically a miniaturized ResNet [33] (§8), as proxy models. A specialized NN is a NN that mimics a larger NN (e.g., Mask R-CNN) on a simplified task, e.g., on a marginal distribution of the larger NN. As specialized NNs predict simpler output, they can run dramatically faster.

BLAZEIT will infer if a specialized NN can be trained from the query specification. For example, to replicate NOSCOPE's binary detection, BLAZEIT would infer that there is a predicate for whether or not there is an object of interest in the frame and train a specialized NN to predict this. Prior work has used specialized NNs for binary detection [31,46], but we extend specialization for aggregation and limit queries.

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

Field	Type	Description
`timestamp`	float	Time stamp
`class`	string	Object class (e.g., bus, car)
`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.

3.2 Limitations

While BLAZEIT can answer a significantly wider range of video queries than prior work, we highlight several limitations.

TMAS. BLAZEIT requires the object detection method to be run over a portion of the data for training specialized NNs and filters as a preprocessing step. Other contemporary systems also require a TMAS [37,46].

Model drift. BLAZEIT targets on the batch analytics setting where the TMAs can be sampled i.i.d. from the data. However, in the streaming setting, where the data distribution may change, BLAZEIT will still provide accuracy guarantees but performance may be reduced. Namely, the accuracy of BLAZEIT's specialized NNs may degrade relative to the target model. As a result, BLAZEIT may execute queries more slowly, but *this will not affect accuracy* (§5). This effect can be mitigated by labeling a portion of new data and monitoring drift or continuous retraining.

Object detection. BLAZEIT depends on the target object detection method and does not support object classes beyond what the method returns, e.g., the pretrained Mask R-CNN [26, 32] can detect cars, but cannot distinguish between sedans and SUVs. However, users can supply UDFs if necessary.

4. FRAMEQL: EXPRESSING COMPLEX SPATIOTEMPORAL VISUAL QUERIES

To address the need for a unifying query language over video analytics, we introduce FRAMEQL, an extension of SQL for querying spatiotemporal information of objects in video. By providing a table-like schema using the standard relational algebra, we enable users familiar with SQL to query videos, whereas implementing these queries manually would require expertise in deep learning, computer vision, and programming.

FRAMEQL is inspired by prior query languages for video analytics [18, 48, 50, 54], but FRAMEQL specifically targets information that can be populated automatically using computer vision methods. We discuss differences in detail at the end of this section.

FRAMEQL data model. FRAMEQL represents videos (possibly compressed in formats such as H.264) as virtual relations, with one relation per video. Each FRAMEQL tuple corresponds to a single object in a frame. Thus, a frame can have zero or more tuples (i.e.,

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

Figure 4: FRAMEQL syntax. As shown, FRAMEQL largely inherits SQL syntax.

zero or more objects), and the same object can have one or more tuples associated with it (i.e., appear in several frames).

We show FRAMEQL's data schema in Table 1. It contains fields relating to the time, location, object class, and object identifier, the box contents, and the features from the object detection method. BLAZEIT can automatically populate `mask`, `class`, and `features` from the object detection method (see Eq. 1), `trackid` from the entity resolution method, and `timestamp` and `content` from the video metadata. Users can override the default object detection and entity resolution methods. 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.[1]

FRAMEQL query format. FRAMEQL allows selection, projection, and aggregation of objects, and, by returning relations, can be composed with standard relational operators. We show the FRAMEQL syntax in Figure 4. FRAMEQL extends SQL in three ways: `GAP`, syntax for specifying an error tolerance (e.g., `ERROR WITHIN`), and `FCOUNT`. Notably, we do not support joins as we do not optimize for joins in this work, but we describe how to extend FRAMEQL with joins in an extended version of this paper [44]. We show FRAMEQL's extensions Table 2; several were taken from BlinkDB [5]. We provide the motivation behind each additional piece of syntax.

First, when the user selects timestamps, the `GAP` keyword ensures that the returned frames are at least `GAP` frames apart. For example, if 10 consecutive frames contain the event and `GAP = 100`, only one frame of the 10 frames would be returned.

Second, as in BlinkDB [5], users may wish to have fast responses to exploratory queries and may tolerate some error. Thus, we allow the user to specify error bounds in the form of maximum absolute error, false positive error, and false negative error, along with a specified confidence level (e.g., Figure 2a). NOSCOPE's pipeline can be replicated with FRAMEQL using these constructs. We choose absolute error bounds in this work as the user may inadvertanely execute a query with 0 records, which would require scanning the entire video (§6).

We also 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. `FCOUNT` 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. Window-based analytics can be done using the existing `GROUP BY` keyword.

FRAMEQL examples. We describe how the some of the example use cases from §2 can be written in FRAMEQL. We assume the video is recorded at 30 fps.

[1] BLAZEIT will inherit any errors from the object detection and entity resolution methods.

Table 2: Additional syntactic elements in FRAMEQL. Some of these were adapted from BlinkDB [5].

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 level
`GAP`	Minimum distance between returned frames

Table 3: A comparison of object detection methods, filters, and speeds. More accurate object detection methods are more expensive. Specialized NNs and simple filters are orders of magnitude more efficient than object detection methods.

Method	mAP	FPS
YOLOv2 [62]	25.4	80
Mask R-CNN [32]	45.2	3
Specialized NN	N/A	35k
Decoding low-resol video	N/A	62k
Color filter	N/A	100k

Figure 2a 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 2b shows how to select frames with at least one bus and at least five cars, which uses the `GAP` keyword to ensure events are a certain time apart. At 30 fps, `GAP 300` corresponds to 10 seconds. Figure 2c shows how to exhaustively select frames with red buses. Here, `redness` and `area` are UDFs, as described in §3. The other example use cases can be answered similarly.

Comparison to prior languages. Prior visual query engines have proposed similar schemas, *but assume that the relation is already populated* [47, 52], i.e., that the data has been created through external means (typically by humans). In contrast, FRAMEQL's relation 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, which is similar to an unmaterialized view. This form of laziness enables a variety of optimizations via query planning.

5. QUERY OPTIMIZER OVERVIEW

Overview. BLAZEIT's primary challenge is executing FRAMEQL queries *efficiently*: recall that object detection is the overwhelming bottleneck (Table 3). To optimize and execute queries, BLAZEIT inspects query contents to see if optimizations can be applied. For example, BLAZEIT cannot optimize aggregation queries without error bounds, but can optimize aggregation queries with a user-specified error tolerance.

BLAZEIT leverages two novel optimizations to reduce the computational cost of object detection, targeting aggregation (§6) and limit queries (§7). 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 (Table 3). Thus, we have found a rule-based optimizer to be sufficient in optimizing FRAMEQL queries.

Both of BLAZEIT's novel optimizations share a key property: they still provide accuracy guarantees despite using potentially inaccurate specialized NNs. Specifically, both optimization will only

speed up query execution and will not affect the accuracy of queries; full details are in §6 and §7.

BLAZEIT also can optimize exhaustive selection queries with predicates by implementing optimizations in prior work, such as using NOSCOPE's specialized NNs as a filter [46, 55]. As this case has been studied, we defer the discussion of BLAZEIT's query optimization for exhaustive selection to an extended paper [44].

BLAZEIT's rule-based optimizer will inspect the query specification to decide which optimizations to apply. First, if the query specification contains an aggregation keyword, e.g., `FCOUNT`, BLAZEIT will apply our novel optimization for fast aggregation. Second, if the query specification contains the `LIMIT` keyword, BLAZEIT will apply our novel optimization for limit queries. Finally, for all other queries, BLAZEIT will default to applying filters similar to NOSCOPE's [46].

Work reuse. In addition to our novel optimizations, BLAZEIT can reuse work by storing the specialized NN model weights and their results. The specialized NNs BLAZEIT uses are small, e.g., < 2 MB, compared to the size of the video.

We describe the intuition, the physical operator(s), its time complexity and correctness, and the operator selection procedure for aggregates (§6) and limit queries (§7) below.

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; see Figure 2a for an example. To provide exact answers, BLAZEIT must call object detection on every frame, which is prohibitively slow. However, if the user specifies an error tolerance, BLAZEIT accelerate query execution using two novel optimizations.

We focus on optimizing counting the number of objects in a frame. BLAZEIT requires training data from the TMAS (§2) of the desired quantity (e.g., number of cars) to leverage specialized NNs. If there is insufficient training data, BLAZEIT will default to random sampling. If there is sufficient training data, BLAZEIT will first train a specialized NN to estimate the statistic: if the specialized NN is accurate enough, BLAZEIT can return the answer directly. Otherwise, BLAZEIT will use specialized NNs to reduce the variance of AQP via control variates [28], requiring fewer samples. We next describe these steps in detail.

Operator Selection. The process above is formalized in Algorithm 1. BLAZEIT will process the TMAS into training data for a specialized NN by materializing labels, i.e., counts. Given these labels, BLAZEIT first determines whether there is sufficient training data ($> 1\%$ of the data has instances of the object; this choice will only affect runtime, not accuracy) 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 bears), BLAZEIT will default to standard random sampling. We use an adaptive sampling algorithm that respects the user's error bound but can terminate early based on the variance of the sample [56].

When there is sufficient training data, BLAZEIT will train a specialized NN and estimate its error rate on the held-out set. If the error is smaller than the specified error at the confidence level, it will then execute the specialized NN on the unseen data and return the answer directly. For specialized NN execution, BLAZEIT will subsample at twice minimum frequency of objects appearing; the minimum frequency is estimated from the TMAS. Sampling at this rate, i.e., the Nyquist rate [57], will ensure that BLAZEIT will sample all objects. As specialized NNs are significantly faster than object detection, this procedure results in much faster execution.

Data: TMAS, 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 TMAS;
  $err \leftarrow$ specialized NN error rate;
  $\tau \leftarrow$ average of specialized NN over unseen video;
  if $P(err < uerr) < conf$ **then**
    return τ;
  else
    $\hat{m} \leftarrow$ result of Equation 2 (control variates);
    return $\hat{m}$;
  end
else
  Return result of random sampling.;
end

Algorithm 1: BLAZEIT's aggregation query procedure. BLAZEIT will use specialized NNs for accelerated query execution via control variates or query rewriting where possible.

When the specialized NN is not accurate enough, it is used as a control variate: a cheap-to-compute auxiliary variable correlated with the true statistic. Control variates can approximate the statistic with fewer samples than naive 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 samples from the video, populating at most a small number of rows for faster execution. Similar to online aggregation [34], we provide absolute error bounds, but the algorithm could be easily modified to give relative error bounds. BLAZEIT uses Empirical Bernstein stopping (EBS) [56], which allows for early termination based on the variance, which is useful for control variates. We specifically use Algorithm 3 in [56]; we give an overview of this algorithm in an extended version of this paper [44].

EBS provides an always valid, near-optimal stopping rule for bounded random variables. EBS is always-valid in the sense that when EBS terminates, it will respect the user's error bound and confidence; the guarantees come from a union bound [56]. EBS is near-optimal in the following sense. Denote the user-defined error and confidence as ϵ and δ. Denote the range of the random variable to be R. EBS will stop within $c \cdot \log\log \frac{1}{\epsilon \cdot |\mu|}$ of any optimal stopping rule that satisfies ϵ and δ. Here, c is a constant and $|\mu|$ is the mean of the random variable.

Query Rewriting via Specialized NNs. 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 [33]) for one epoch with a fixed learning rate of 0.1.

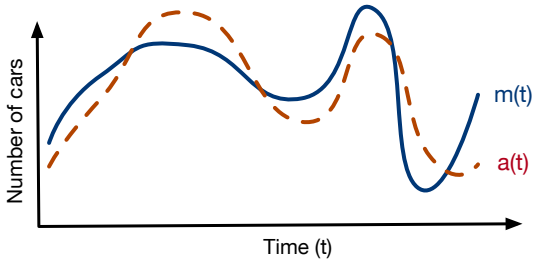


Figure 5: Schematic of control variates. Here, $a(t)$ is the result of the specialized NN and $m(t)$ is the result of object detection. a is cheap to compute, but possibly inaccurate. Nonetheless, $\hat{m} = m + c \cdot (a - \mathbb{E}[a])$ has lower variance than m; thus we can use a to compute $\mathbb{E}[m]$ with fewer samples from m.

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 of using specialized NNs while still guaranteeing accuracy. In particular, we adapt the method of control variates [28] 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 [42, 64]) 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 m and we have access to an auxiliary variable a. The desiderata for a are that: 1) a is cheaply computable, 2) a is correlated with m (see time complexity). We further assume we can compute $\mathbb{E}[a] = \alpha$ and $\text{Var}(a)$ exactly. Then,

$$\hat{m} = m + c \cdot (a - \alpha) \tag{2}$$

is an unbiased estimator of m for any choice of c [28]. The optimal choice of c is $c = -\frac{\text{Cov}(m,a)}{\text{Var}(a)}$ and using this choice of c gives $\text{Var}(\hat{m}) = (1 - \text{Corr}(m,a)^2)\text{Var}(m)$. As an example, suppose $a = m$. Then, $\hat{m} = m + c(m - \mathbb{E}[m]) = \mathbb{E}[m]$ and $\text{Var}(\hat{m}) = 0$.

This formulation works for arbitrary a, but choices where a is correlated with m give the best results. As we show in §9.2, specialized NNs 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; we show a schematic in Figure 5. 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 NN would exactly match 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 a 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 $\text{Cov}(m, a)$ at every round.

Aggregation with query predicates. A user might issue an aggregation query that contains predicates such as filtering for large red buses (see Figure 3). In this case, BLAZEIT will execute a similar procedure above, but first applying the predicates to the training data. The key difference is that in cases where there is not enough training data, BLAZEIT will instead generate a specialized NN to count the most selective set of predicates that contains enough data.

For example, consider a query that counts the number of large red buses. If there is not enough data to train a specialized NN that counts the number of large red buses, BLAZEIT will instead train a specialized NN that counts the number of large buses (or red buses, depending on the training data). If there is no training data for the quantity of interest, BLAZEIT will default to standard sampling.

As control variates only requires that the proxy variable, i.e., the specialized NN in this case, be *correlated* with the statistic of interest, BLAZEIT will return a correct answer even if it trains a specialized NN that does not directly predict the statistic of interest.

Correctness. The work in [56] proves that EBS is an always valid, near-optimal stopping rule. Briefly, EBS maintains an upper and lower bound of the estimate that always respects the confidence interval and terminates when the error bound is met given the range of the data. We estimate the range from the TMAS, which we empirically show does not affect the confidence intervals in Appendix D. Furthermore, while video is temporally correlated, we assume all the video is present, namely the batch setting. As a result, shuffling the data will result in i.i.d. samples. Control variates are an unbiased estimator for the statistic of interest [28], so standard proofs of correctness apply to control variates.

Query rewriting using specialized NNs will respect the requested error bound and confidence level under the assumption of no model drift (see §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 level and the given video). Denote the standard deviation of random sampling 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 increases (which typically happens when specialized NNs are more accurate or when σ_a is large), control variates give larger speedups.

While σ_a and σ_c depend on the query, we empirically show in §9 that control variates and query rewriting are beneficial.

7. OPTIMIZING LIMIT QUERIES

Overview. In cardinality-limited queries, the user is interested in finding a limited number of events, (e.g., 10 events of a bus and five cars, see Figure 2b), typically for manual inspection. Limit queries are especially helpful for rare events. To answer these queries, BLAZEIT could perform object detection over every frame to search for the events. However, if the events occurs infrequently, naive methods of random sampling or sequential scans of the video can be prohibitively slow (e.g., at 30 fps, an event that occurs once every 30 minutes corresponds to a rate of 1.9×10^{-5}).

Our key intuition is to bias the search towards frames that likely contain the event. We use specialized NNs for biased sampling, in a similar vein to techniques from the rare-event simulation literature [43]. 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}) [43].

Physical operator and selection. BLAZEIT currently supports limit 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:

- If there are 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 in prior work [46] (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 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. The training data for the specialized NN is generated in the same way for aggregation queries (§6). 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 predict counts 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. BLAZEIT use the sum of the probability of the frame having at least one bus and at least five cars as its signal. 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 predict each object class separately (e.g., instead of jointly predicting "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 the `GAP` keyword, once an event is found, the surrounding `GAP` frames are ignored.

Limit queries with multiple predicates. As with aggregation queries, a user might issue a limit query with predicates. If there is sufficient training data in the TMAS, BLAZEIT can execute the procedure above. If there is not sufficient training data, BLAZEIT will train a specialized NN to search for the most selective set of predicates that contains enough data in a similar fashion to generating an aggregation specialized NN.

Correctness. BLAZEIT performs object detection on all sampled frames, so it always returns an exact answer. All frames will be exhaustively searched if there are fewer events than the number requested, which will also be exact.

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. Once an event is found, the `GAP` frames around the event can be ignored, but this is negligible in practice so 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.

While using specialized NNs to bias the search does not guarantee faster runtime, we show in §9 that it empirically can reduce the number of frames considered.

8. IMPLEMENTATION

We implemented our prototype of BLAZEIT in Python 3.5 for the control plane (the deep learning frameworks we use for object detection require Python) and, for efficiency purposes, we implement the non-NN filters in C++. We use PyTorch v1.0 for the training and evaluation of specialized NNs. For object detection, we use FGFA [73] using MXNet v1.2 and Mask R-CNN [32] using the Detectron framework [26] 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.

BLAZEIT uses a Fluent DSL written in Python to specify FRAMEQL queries. The cost of storing and materializing the processed data is negligible, so we use Pandas for processing tuples.

Video ingestion. BLAZEIT loads the video and resizes the frames to the appropriate size for each NN (65×65 for specialized NNs, short side of 600 pixels for object detection methods), and normalizes the pixel values appropriately.

Specialized NN training. We train the specialized NNs using PyTorch v1.0. Video are ingested and resized to 65×65 pixels and normalized using standard ImageNet normalization [33]. Standard cross-entropy loss is used for training, with a batch size of 16. We use SGD with a momentum of 0.9. Our specialized NNs use a "tiny ResNet" architecture, a modified version of the standard ResNet architecture [33], which has 10 layers and a starting filter size of 16, for all query types. As this work focuses on exploratory queries, we choose tiny ResNet as a good default and show that it performs better than or on par with the NNs used in [46].

Identifying objects across frames. Our default for computing `trackid` uses motion IOU [73]. 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.

9. EVALUATION

We evaluated BLAZEIT on a variety of aggregation and limit FRAMEQL queries on real-world video streams. We show that:

1. BLAZEIT achieves up to a 14× speedup over AQP on aggregation queries (§9.2).
2. BLAZEIT achieves up to an 83× speedup compared to the next best method for video limit queries (§9.3).

9.1 Experimental Setup

Evaluation queries and videos. We evaluated BLAZEIT on six videos shown in Table 4, which were scraped from YouTube. `taipei`, `night-street`, `amsterdam`, and `archie` are widely used in video analytics systems [9,37,40,46,70] and we collected two other streams. We only considered 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].

Table 4: Video streams and object labels queried in our evaluation. We show the data from the test set, as the data from the test set will influence the runtime of the baselines and BLAZEIT.

Video name	Object	Occupancy	Avg. duration of object in scene	Distinct count	Resol.	FPS	# Eval frames	Length (hrs)	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	0.8
rialto	boat	89.9%	10.7s	5969	720p	30	866k	24	Mask	0.8
grand-canal	boat	57.7%	9.50s	1849	1080p	60	1300k	18	Mask	0.8
amsterdam	car	44.7%	7.88s	3096	720p	30	1188k	33	Mask	0.8
archie	car	51.8%	0.30s	90088	2160p	30	1188k	33	Mask	0.8

We evaluate on queries similar to Figure 2, in which the class and video were changed.

Target object detection methods. For each video, we used a pretrained object detection method as the target object detection method, as pretrained NNs do not require collecting additional data or training: collecting data and training is difficult for non-experts. We selected between Mask R-CNN [32] pretrained on MS-COCO [53], FGFA [73] pretrained on ImageNet-Vid [65], and YOLOv2 [62] pretrained on MS-COCO.

We labeled part of each video using Mask R-CNN [32], FGFA [73], and YOLOv2 [62], and manually selected the most accurate method for each video. Mask R-CNN and FGFA are significantly more accurate than YOLOv2, so we did not select YOLOv2 for any video. The chosen object detection method per video was used for all queries for that video.

In timing the naive baseline, we only included the GPU compute time and exclude the time to process the video and convert tuples to FRAMEQL format, as object detection is the overwhelming computational cost.

Data preprocessing. The literature reports that state-of-the-art object detection methods still suffer in performance for small objects [32, 73]. Thus, we only considered 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 selected confidence thresholds for each video and object class for when to consider an object present (Table 4).

Evaluation metrics. We computed all accuracy metrics with respect to the object detection method, i.e., we treated the object detection method as ground truth. For aggregation queries, we report the absolute error. For limit queries, we guarantee only true positives are returned, thus we only report throughput.

We have found that modern object detection methods can be accurate at the frame level. Thus, we considered accuracy at the *frame level*, in contrast to to the one-second binning that is used in [46] to mitigate label flickering for NOSCOPE.

We measured throughput by timing the complete end-to-end system excluding the time taken to decode video, as is standard [46, 55]. We assume the TMAS is computed offline once, so we excluded the time to generate the TMAS. Unlike in [46], we also show runtime numbers *when the training time of the specialized NN is included*. 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 performed our experiments on a server with a single NVIDIA Tesla P100 GPU and two Intel Xeon E5-2690v4 CPUs (56 threads). The system has 504 GB of RAM.

9.1.1 Binary Oracle Configuration

Many prior visual analytics systems answer binary classification queries, including NOSCOPE, TAHOMA, and probablistic predicates [37,46,55] which are the closest systems to BLAZEIT. These systems cannot directly answer queries in the form of aggregate or limit queries for multiple instances of an object or objects.

As binary classification is not directly applicable to the tasks we consider, where relevant, we compared against a *binary 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 existing systems. We describe how the binary oracle can be used to answer each type of query.

Aggregates. Binary oracles cannot distinguish between one and several objects, so object detection must be performed on every frame with an object to identify the individual objects. Thus, counting cars in taipei would require performing object detection on 64.4% of the frames, i.e., the occupancy rate.

Cardinality-limited queries. As above, a binary oracle 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, a binary oracle 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.

9.2 Aggregate Queries

We evaluated BLAZEIT on six aggregate queries across six videos. The queries are similar to the query in Figure 2a, with the video and object class changed. We ran five variants of each query:

- Naive: we performed object detection on every frame.
- Binary oracle: we performed object detection on every frame with the object class present.
- Naive AQP: we randomly sampled from the video.
- BLAZEIT: we used specialized NNs and control variates for efficient sampling.
- BLAZEIT (no train): we excluded the training time.

There are two qualitatively different execution modes: 1) where BLAZEIT rewrites the query using a specialized NN and 2) where BLAZEIT samples using specialized NNs as control variates (§6). We analyzed these cases separately.

Query rewriting via specialized NNs. We evaluated the runtime and accuracy of specialized NNs when the query can be rewritten by using a specialized NN. We ran each query with a target error rate of 0.1 and a confidence level of 95%. We show the average of three runs. Query rewriting was unable to achieve this accuracy for archie, so we excluded it. However, we show below that specialized NNs can be used as a control variate even in this case.

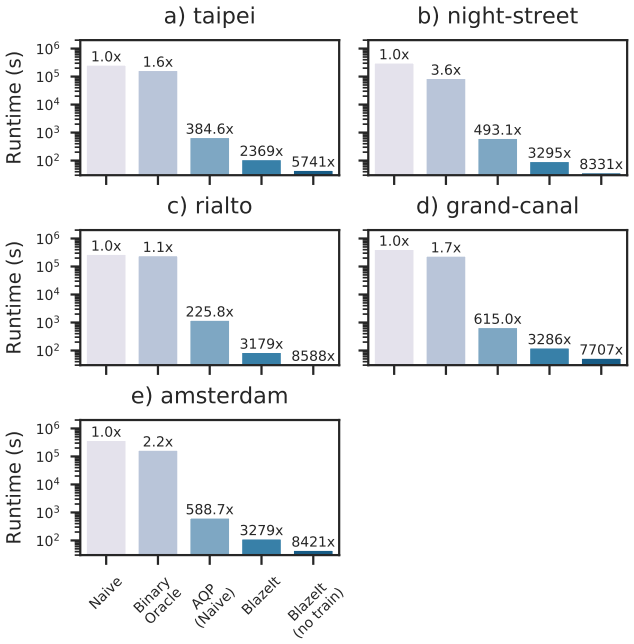


Figure 6: End-to-end runtime of aggregate queries where BLAZEIT rewrote the query with a specialized network, measured in seconds (log scale). BLAZEIT outperforms all baselines. All queries targeted $\epsilon = 0.1$.

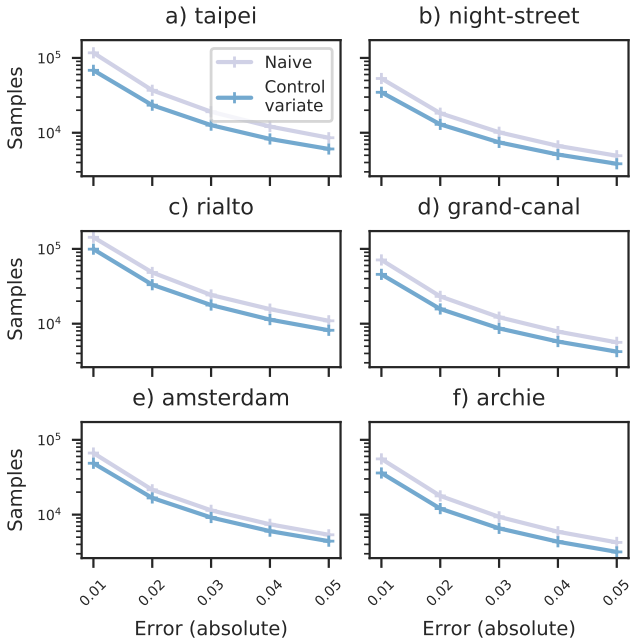


Figure 7: Sample complexity of random sampling and BLAZEIT with control variates. Control variates via specialized NNs consistently outperforms standard random sampling. Note the y-axis is on a log scale.

Table 5: Average error of 3 runs of query-rewriting using a specialized NN for counting. These videos stayed within $\epsilon = 0.1$.

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

Table 6: Estimated and true counts for specialized NNs run on two different days of video. In parentheses are the day of video.

Video	Pred (1)	Actual (1)	Pred (2)	Actual (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

As shown in Figure 6, BLAZEIT can outperform naive AQP by up to 14× even when including the training time and time to compute thresholds, which the binary oracle does not include. The binary oracle baseline does not perform well when the video has many objects of interest (e.g., rialto).

While specialized NNs do not provide error guarantees, we show that the absolute error stays within the 0.1 for the given videos in Table 5. This shows that specialized NNs can be used for query rewriting while respecting the user's error bounds.

Sampling and control variates. We evaluated the runtime and accuracy of sampling with specialized NNs as a control variate. Because of the high computational cost of running object detection, we ran the object detection method once and recorded the results. The run times in this section are estimated from the number of object detection invocations.

We targeted error rates of 0.01, 0.02, 0.03, 0.04, and 0.05 with a confidence level of 95%. We averaged the number of samples for each error level over 100 runs.

As shown in Figure 7, using specialized NNs as a control variate can deliver up to a 1.7× reduction in sample complexity. As predicted by theory, the reduction in variance depends on the correlation between the specialized NNs and the object detection methods. Specifically, as the correlation coefficient increases, the sample complexity decreases.

Sampling with predicates. We evaluated the runtime of BLAZEIT on aggregation queries with predicates. We evaluated on one query per video and counted the number of objects with a given color and at least a given size; full query details are give in an extended version of this paper [44]. We targeted an error rate of 0.001.

As shown in Figure 8, using specialized NNs as control variates can deliver up to a 1.5× speedup compared to naive AQP. While the absolute runtimes vary depending on the difficulty of the query, the relative gain of BLAZEIT's control variates only depends on the reduction in variance. Finally, we note the gains are lower compared to queries with predicates as there is less training data.

Specialized NNs do not learn the average. Specialized NNs may perform well by learning the average number of cars. To demonstrate that they do not, we swapped 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 6. 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.

9.3 Cardinality-limited Queries

We evaluated BLAZEIT on limit queries, in which frames of interest are returned to the user, up to the requested number of frames. The queries are similar to the query in Figure 2b. We show in Table 7 the query details and the number of instances of each query. If the user queries more than the maximum number of instances, BLAZEIT must inspect every frame. Thus, we chose queries with at least 10 instances of the query.

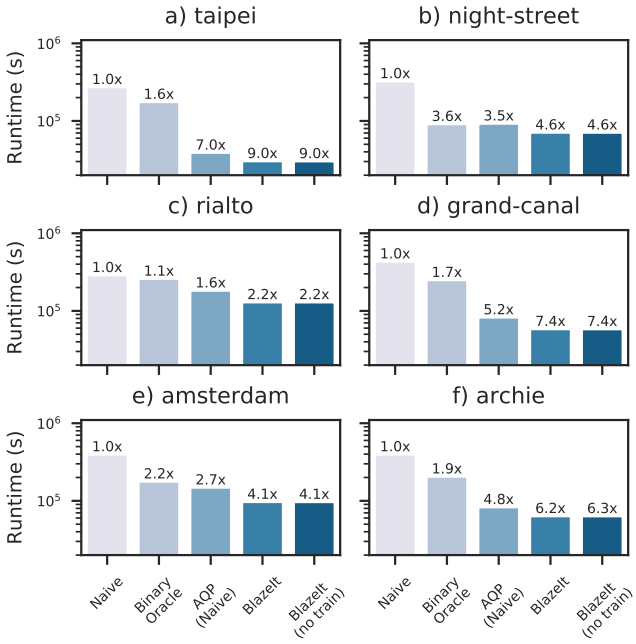


Figure 8: Runtime of BLAZEIT and baselines for aggregation queries with predicates. Note the y-axis is on a log scale. As shown, BLAZEIT consistently outperforms naive random sampling.

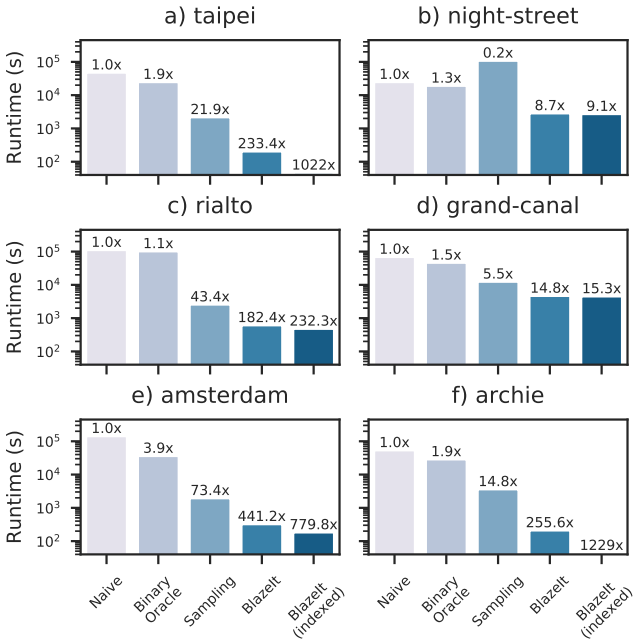


Figure 9: End-to-end runtime of baselines and BLAZEIT on limit queries; BLAZEIT outperforms all baselines. The y-axis is log-scaled. All queries looked for 10 events.

Table 7: Query details and number of instances. We selected rare events with at least 10 instances.

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

BLAZEIT will only return true positives for limit queries (§7), thus we only report the runtime. Additionally, if we suppose that the videos are indexed with the output of the specialized NNs, we can simply query the frames using information from the index. This scenario might occur when the user executed an aggregate query as above. Thus, we additionally report sample complexity.

We ran the following variants:

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

Single object class. Figure 9 shows that BLAZEIT can achieve over a 1000× speedup compared to baselines. We see that the baselines do poorly in finding rare objects, where BLAZEIT's specialized NNs can serve as a high-fidelity signal.

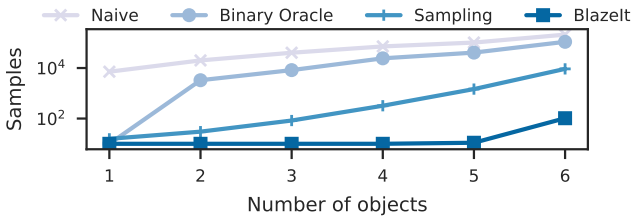


Figure 10: Sample complexity of baselines and BLAZEIT when searching for at least N cars in taipei; BLAZEIT outperforms all baselines. Note the y-axis is on a log-scale. All queries searched for 10 events.

We also varied the number of cars in taipei to see if BLAZEIT could also search for common objects. As shown in Figure 10, the sample complexity increases as the number of cars increases for both the naive method and the binary oracle. 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 tested BLAZEIT on multiple object classes by searching for at least one bus and at least five cars in taipei. There are 63 instances in the test set.

As shown in Figure 11, BLAZEIT outperforms the naive baseline by up to 966×. Searching for multiple object classes is favorable for the binary oracle, as it becomes more selective. Nonetheless, BLAZEIT significantly outperforms the binary oracle, giving up to a 81× performance increase.

Additionally, we show the sample complexity as a function of the LIMIT in Figure 12 of BLAZEIT and the baselines, for taipei. We see that BLAZEIT can be up to orders of magnitude more sample efficient over both the naive baseline and the binary oracle.

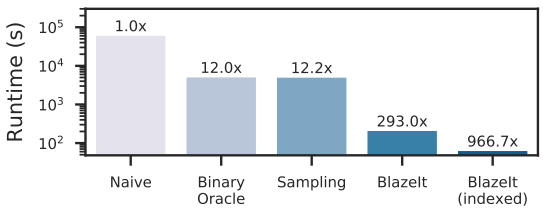


Figure 11: End-to-end runtime of baselines and BLAZEIT on finding at least one bus and at least five cars in `taipei`; BLAZEIT outperforms all baselines. Note the y-axis is on a log scale.

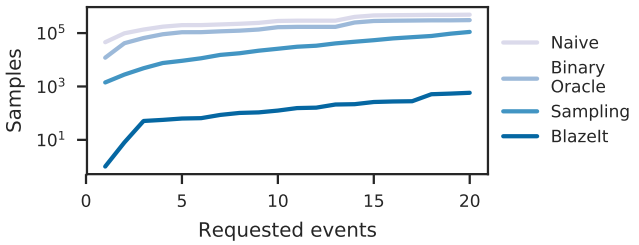


Figure 12: Sample complexity of BLAZEIT and baselines when searching for at least one bus and at least five cars in `taipei`; BLAZEIT outperforms all baselines. The x-axis is the number of requested frames. Note the y-axis is on a log scale.

Limit queries with predicates. We evaluated BLAZEIT on limit queries with predicates by searching for objects with a specified color and at least a specified size. We present the full query details and statistics in an extended version of this paper [44].

As shown in Figure 13, BLAZEIT outperforms all baselines by up to 300×, even when including the proxy model training time. BLAZEIT especially outperforms baselines on queries that have few matches, as random sampling and NOSCOPE will perform poorly in these settings.

9.4 Specialized Neural Networks

Effect of NN type. In this work, we used 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 NN [46], parameterized by 32 base filters, 32 dense neurons, and 4 layers.

We used TRN10 and the NOSCOPE NN on limit queries for each of the videos and computed the number of samples required to find the requested number of events in Table 7. As shown in Figure 14, TRN10 requires significantly fewer samples compared to the NOSCOPE NN on all videos.

We additionally used TRN10 and the NOSCOPE NN 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 15, TRN10 typically matches or beats the NOSCOPE NN, except for `night-street`.

Effect of training data. To see the effect of the amount of training data on aggregate query performance, we plotted the error of the specialized NNs on the aggregation queries used in Section 9.2. We show results in Figure 16. As shown, the error tends to decrease until around 150,000 training examples and levels off or increases, potentially due to overfitting.

Performance benchmarks. We plotted the performance of tiny ResNets as the depth, width, and resolution of the network and

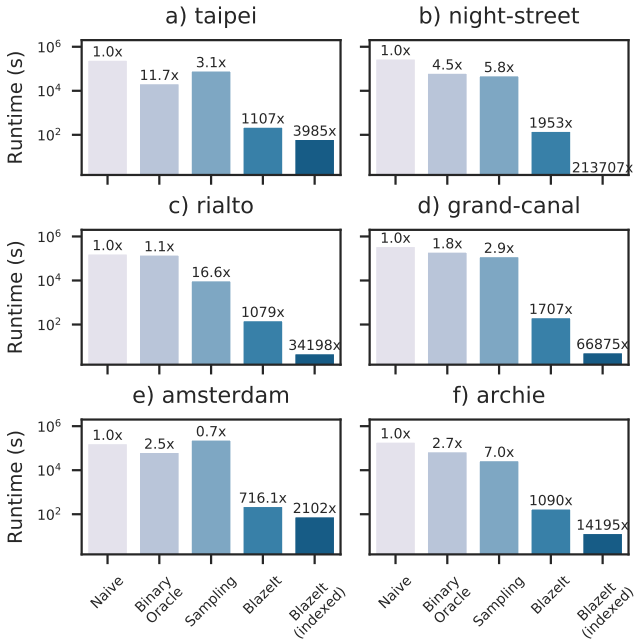


Figure 13: Runtime of BLAZEIT and baselines on limit queries with predicates. BLAZEIT's outperforms all baselines, even when including the training time of the proxy model. BLAZEIT especially outperforms baselines when the selectivity is high: random sampling will perform especially poorly on rare events.

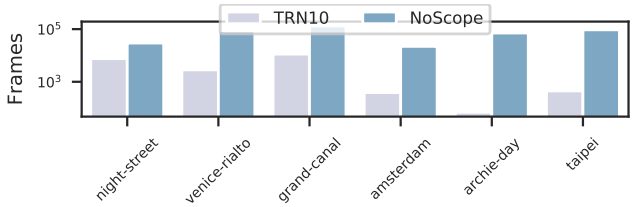


Figure 14: Number of samples to find 10 of the requested objects for each query, using TRN10 or a representative NOSCOPE NN. As shown, TRN10 significantly outperforms the NOSCOPE NN on all videos. The y-axis is on a log-scale. Average of 3 runs.

inputs varied. As shown in Figure 17, the throughput of the tiny ResNets decreases linearly with depth and width. The throughput generally decreases with the square of the resolution, but reduces further if the maximum batch size that the GPU can fit decreases.

10. RELATED WORK

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

AQP. 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 [34]. Prior work has leveraged various sampling methods [4, 11], histograms [3, 15, 27, 60], and sketches [10, 35, 41].

The key different in BLAZEIT is difference in cost of tuple materialization: materializing a tuple for video analytics (i.e., executing object detection) is orders of magnitude more expensive than in standard databases. To address this challenge, we introduce a new form of variance reduction in the form of control variates [28] via specialized NNs. This form of variance reduction, and others

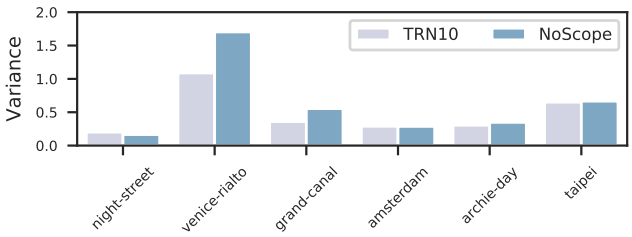


Figure 15: Variance of the control variates estimator when using TRN10 or a representative NoScope NN (lower is better). As shown, TRN10 typically matches or beats the NoScope NN, except for `night-street`.

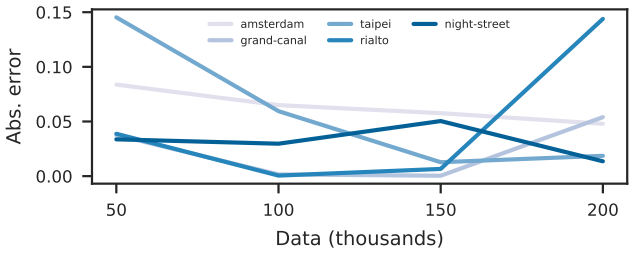


Figure 16: Effect of the amount of data on the error of specialized NNs on the aggregation queries used in Section 9.2. As shown, the error tends to decrease until 150k training examples.

involving auxiliary variables, does not apply in a traditional relational database due to the cost imbalance.

Visual data management. Visual data management has aimed to organize and query visual data, starting from systems such as Chabot [58] and QBIC [21]. These systems were followed by a range of "multimedia" database for storing [8, 51], querying [7, 49, 59], and managing [25, 39, 71] video data. The literature also contains many proposals for query languages for visual data [17, 38, 54]; we discuss how FrameQL differs from these languages in an extended version of this paper [44].

Many of these systems and languages use classic computer vision techniques such as *low-level* image features (e.g., color) 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. In this work, we explicitly choose to extend SQL in FrameQL and focus on how these fields can be *automatically populated* rather than the syntax.

Modern video analytics. Systems builders have created video analytics systems, e.g., NoScope [46], a highly tuned pipeline for binary detection: it returns the presence or absence of a particular object class in video. Other systems, e.g., Focus [37] and Tahoma [6], also optimize 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 optimize the training time of specialized NNs. In BlazeIt, we extend specialization and present novel optimizations for aggregation and limit queries, which these systems do not support.

Other contemporary work use filters with a false negative rate (called probabilistic predicates) that are automatically learned from a hold-out set [55]. These could be incorporated into BlazeIt for selection queries.

Other systems aim to reduce latency of live queries (e.g., VideoStorm [72]) or increase the throughput of batch analytics queries (e.g., Scanner [61]) that are pre-defined *as a computation graph*. As the computation is specified as a black-box, these systems do not have access to the semantics of the computation to perform certain optimizations, such as in BlazeIt. In BlazeIt, we introduce FrameQL and an optimizer that can infer optimizations from the given query. Additionally, BlazeIt could be integrated with VideoStorm for live analytics or Scanner for scale-out.

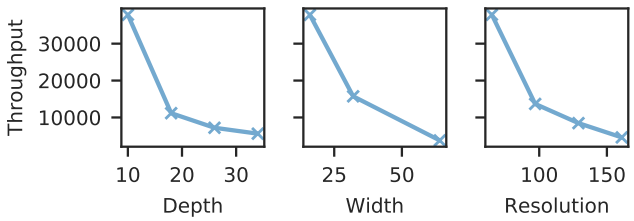


Figure 17: Effect of the width, depth, and input resolution on the throughput of the tiny ResNet architecture. The throughput is proportional to the inverse of the width and the depth, and generally proportional to the inverse of the square of the input resolution.

We presented a preliminary version of BlazeIt as a non-archival demonstration [45].

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

First, a large body of work changes the NN architecture or weights for improved inference efficiency, that preserve the full generality of these NNs. Model compression uses a variety of techniques from pruning [30] to compressing [12] weights from the original NN, which can be amenable to hardware acceleration [29]. Model distillation uses a large NN to train a smaller NN [36]. These methods are largely orthogonal to BlazeIt, and reducing the cost of object detection would also improve BlazeIt's runtime.

Second, specialization [46, 67] aims to improve inference speeds by training a small NN to mimic a larger NN *on a reduced task*. Specialization has typically been applied in *specific* pipelines, e.g., for binary detection. In BlazeIt, we extend specialization to counting and multi-class classification. Further, we show to how use specialized NNs as control variates and for limit queries.

11. CONCLUSIONS

Querying video for semantic information has become possible with advances in computer vision. However, these NNs run up to 10× slower than real-time and requires complex programming with low-level libraries to deploy. In response, we present BlazeIt, a optimizing video analytics system with a declarative language, FrameQL. We introduce two novel optimizations for aggregation and limit queries, which are not supported by prior work. These techniques can run orders of magnitude faster than baselines while retaining accuracy guarantees, despite potentially inaccurate specialized NNs. These results suggest that new classes of queries can be answered over large video datasets with orders of magnitude lower computational cost.

Acknowledgements

This research was supported in part by affiliate members and other supporters of the Stanford DAWN project—Ant Financial, Facebook, Google, Infosys, Intel, NEC, SAP, Teradata, and VMware—as well as Toyota Research Institute, Keysight Technologies, Amazon Web Services, Cisco, and the NSF under CAREER grant CNS-1651570. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

12. 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 *PVLDB*, pages 754–757, 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 *SIGMOD*, 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 *EuroSys*, pages 29–42. ACM, 2013.

[6] M. R. Anderson, M. Cafarella, T. F. Wenisch, and G. Ros. Predicate optimization for a visual analytics database. *ICDE*, 2019.

[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 *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 *International Conference on Multimedia*, pages 267–272. ACM, 1993.

[9] C. Canel, T. Kim, G. Zhou, C. Li, H. Lim, D. Andersen, M. Kaminsky, and S. Dulloor. Scaling video analytics on constrained edge nodes. *SysML*, 2019.

[10] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *ICALP*, pages 693–703. Springer, 2002.

[11] S. Chaudhuri, G. Das, and V. Narasayya. Optimized stratified sampling for approximate query processing. *TODS*, 2007.

[12] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *ICML*, 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 *SIGMOD*, 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 *FQAS*, 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*, 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. *TPAMI*, 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, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*. IEEE, 2012.

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

[25] S. Gibbs, C. Breiteneder, and D. Tsichritzis. Audio/video databases: An object-oriented approach. In *ICDE*. IEEE, 1993.

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

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

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

[29] 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 *ISCA*, pages 243–254. IEEE, 2016.

[30] 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.

[31] 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 *MobiSys*, pages 123–136. ACM, 2016.

[32] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *ICCV*, pages 2980–2988. IEEE, 2017.

[33] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

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

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

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

[37] 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. *OSDI*, 2018.

[38] E. Hwang and V. Subrahmanian. Querying video libraries. *Journal of Visual Communication and Image Representation*, 1996.

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

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

[41] C. Jin, W. Qian, C. Sha, J. X. Yu, and A. Zhou. Dynamically maintaining frequent items over a data stream. In *CIKM*, pages 287–294. ACM, 2003.

[42] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *NIPS*, pages 315–323, 2013.

[43] 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.

[44] D. Kang, P. Bailis, and M. Zaharia. Blazeit: Optimizing declarative aggregation and limit queries for neural network-based video analytics. *arXiv preprint arXiv:arXiv:1805.01046*, 2019.

[45] D. Kang, P. Bailis, and M. Zaharia. Challenges and opportunities in dnn-based video analytics: A demonstration of the blazeit video query engine. CIDR, 2019.

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

[47] T. C. Kuo and A. L. Chen. A content-based query language for video databases. In *ICMCS*, pages 209–214. IEEE, 1996.

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

[49] M. La Cascia and E. Ardizzone. Jacob: Just a content-based query system for video databases. In *ICASSP*. IEEE, 1996.

[50] T.-L. Le, M. Thonnat, A. Boucher, and F. Brémond. A query language combining object features and semantic events for surveillance video retrieval. In *MMM*. Springer, 2008.

[51] 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.

[52] J. Z. Li, M. T. Ozsu, D. Szafron, and V. Oria. Moql: A multimedia object query language. In *MIPR*, pages 19–28, 1997.

[53] 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 *ECCV*, pages 740–755. Springer, 2014.

[54] C. Lu, M. Liu, and Z. Wu. Svql: A sql extended query language for video databases. *IJDTA*, 2015.

[55] Y. Lu, A. Chowdhery, S. Kandula, and S. Chaudhuri. Accelerating machine learning inference with probabilistic predicates. In *SIGMOD*, pages 1493–1508. ACM, 2018.

[56] V. Mnih, C. Szepesvári, and J.-Y. Audibert. Empirical bernstein stopping. In *Proceedings of the 25th international conference on Machine learning*, pages 672–679. ACM, 2008.

[57] H. Nyquist. Certain topics in telegraph transmission theory. *Transactions of the American Institute of Electrical Engineers*, 47(2):617–644, 1928.

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

[59] 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.

[60] G. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. *SIGMOD*, 1984.

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

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

[63] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

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

[65] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[66] 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 *AVSS*. IEEE, 2007.

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

[68] H. Shen, S. Han, M. Philipose, and A. Krishnamurthy. Fast video classification via adaptive cascading of deep models. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3646–3654, 2017.

[69] X. Sun, L. Muñoz, and R. Horowitz. Highway traffic state estimation using improved mixture kalman filters for effective ramp metering control. In *IEEE CDC*, volume 6, pages 6333–6338. IEEE, 2003.

[70] T. Xu, L. M. Botelho, and F. X. Lin. Vstore: A data store for analytics on large videos. In *Proceedings of the Fourteenth EuroSys Conference 2019*, page 16. ACM, 2019.

[71] A. Yoshitaka and T. Ichikawa. A survey on content-based retrieval for multimedia databases. *TKDE*.

[72] 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.

[73] 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.