Accelerating Aggregation Queries on Unstructured Streams of Data

Matthew Russo
Stanford University
russom@stanford.edu

Tatsunori Hashimoto
Stanford University
thashim@stanford.edu

Daniel Kang
University of Illinois
Urbana-Champaign
ddkang@illinois.edu

Yi Sun
University of Chicago
yi.sun@uchicago.edu

Matei Zaharia
Stanford University
matei@cs.stanford.edu

ABSTRACT

Analysts and scientists are interested in querying streams of video, audio, and text to extract quantitative insights. For example, an urban planner may wish to measure congestion by querying the live feed from a traffic camera. Prior work has used deep neural networks (DNNs) to answer such queries in the batch setting. However, much of this work is not suited for the streaming setting because it requires access to the entire dataset before a query can be submitted or is specific to video. Thus, to the best of our knowledge, no prior work addresses the problem of efficiently answering queries over multiple modalities of streams.

In this work we propose InQuest, a system for accelerating aggregation queries on unstructured streams of data with statistical guarantees on query accuracy. InQuest leverages inexpensive approximation models ("proxies") and sampling techniques to limit the execution of an expensive high-precision model (an "oracle") to a subset of the stream. It then uses the oracle predictions to compute an approximate query answer in real-time. We theoretically analyzed InQuest and show that the expected error of its query estimates converges on stationary streams at a rate inversely proportional to the oracle budget. We evaluated our algorithm on six real-world video and text datasets and show that InQuest achieves the same root mean squared error (RMSE) as two streaming baselines with up to 5.0x fewer oracle invocations. We further show that InQuest can achieve up to 1.9x lower RMSE at a fixed number of oracle invocations than a state-of-the-art batch setting algorithm.

PVLDB Reference Format:
Matthew Russo, Tatsunori Hashimoto, Daniel Kang, Yi Sun, and Matei Zaharia. Accelerating Aggregation Queries on Unstructured Streams of Data. PVLDB, 16(11): 2897 - 2910, 2023.
doi:10.14778/3611479.3611496

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/stanford-futuredata/InQuest.

Proceedings of the VLDB Endowment, Vol. 16, No. 11 ISSN 2150-8097.
doi:10.14778/3611479.3611496

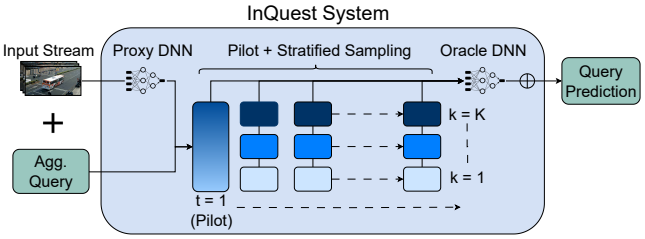

Figure 1: System diagram for InQuest. The user provides an input stream (e.g., a video stream), an aggregation query, a proxy model, and an oracle. InQuest processes the stream with the proxy and performs pilot sampling to compute K initial strata. InQuest applies the oracle to samples drawn from each strata and computes the query prediction.

1 INTRODUCTION

Unstructured streams of data, i.e., streams of data without a well-defined schema (video, audio, and text), are increasingly prevalent. In November 2022, the streaming platform Twitch averaged more than 2 million hours of live video per day [44], while microblogging platforms like Twitter can process 500 million tweets daily [43]. Furthermore, streams are increasingly being used to address real-world problems. As an example, a consortium of universities working with the U.S. Forest Service has deployed hundreds of cameras in the western United States to locate wildfires quickly [3].

Analysts would like to be able to query these streams of data to extract quantitative insights at minimal cost. For example, a social scientist may wish to quantify the sentiment of a Twitter feed during a presidential debate. The researcher may also want to filter for the subset of tweets mentioning a specific candidate.

Prior work has leveraged deep neural networks (DNNs) to execute queries over large datasets of unstructured data [4, 17, 20, 21, 24, 25, 30, 40, 47, 48]. For example, the social scientist might use a BERT model [14] to execute the following query:

```
SELECT COUNT(positive(tweet)) FROM twitter
WHERE mentions_candidate(tweet)
```

The DNN would determine whether the tweet satisfied the query predicate—in this case by mentioning the candidate. It would also compute the statistic of interest, i.e., the sentiment of the tweet.

One limitation of using DNNs for query processing is that executing large DNNs exhaustively over most real-world datasets can

be prohibitively expensive. As a result, recent work has focused on accelerating DNN-based queries [4, 21, 24–27, 35]. One common approach involves filtering (or sub-sampling) records with inexpensive approximation models ("proxies") and then applying high-precision models ("oracles") to extract statistics of interest.

Unfortunately, many of these systems are designed to work in the batch setting and cannot easily be adapted to answer queries over streams. For example, NoScope [25] and Tahoma [4] would need to buffer the entire stream in order to train and validate specialized DNNs and model cascades before answering a query. This limits their ability to answer queries over streams in real-time. Other systems can operate in the streaming setting but are limited to processing a specific modality of data such as video [32]. Thus, to the best of our knowledge, there is currently no system for efficiently processing aggregation queries over multiple modalities of large unstructured streams of data.

In this work we propose and analyze InQuest, a system to accelerate aggregation queries over unstructured streams of data with statistical guarantees on query accuracy. By design, InQuest uses stratified sampling [38] to (1) compute precise query estimates with better sample efficiency than uniform sampling and (2) provide standard frequentist bounds for our query estimates. InQuest takes a stream, an aggregation query, a proxy, and an oracle as input. The user also specifies a tumbling window [16] (i.e., a "segment" length) as well as an oracle budget per segment. As illustrated in Figure 1, InQuest processes the stream segment-by-segment while producing a query estimate in real-time. For each segment, InQuest separates records in the stream into disjoint strata based on their proxy estimates. It then runs the oracle on records sampled from each stratum. Finally, InQuest computes an estimate of the query answer based on the oracle's predictions on the sampled frames.

Performing stratified sampling over streams presents InQuest with multiple challenges. First, InQuest must determine how to stratify the stream records. It then must decide how best to allocate its sampling budget across these strata. Finally, in order to compute unbiased estimates for each stratum, InQuest must draw an unbiased sample from the stream without knowing ahead of time how many records will fall in each stratum. InQuest overcomes these challenges through the use of sampling techniques which we define and provide intuition for in Section 3.1.

We analyze InQuest from a theoretical perspective and show that its allocation strategy and expected error converge at quantitative rates. We first derive the per-segment optimal allocation of our sampling budget assuming perfect knowledge of quantities such as the stratum standard deviations and predicate positive rates. We then show that InQuest's per-segment sample allocation strategy converges to the optimal allocation at a quantitative rate on stationary streams of data. We further show that the expected error of InQuest's estimator converges to zero at a rate inversely proportional to the size of our sampling budget on such streams.

We evaluate InQuest on six real-world video and text datasets and compare it against two streaming baselines—uniform sampling and stratified sampling with fixed strata and fixed sample allocations. We also compare it against ABae [27], a state-of-the-art algorithm for the batch setting which provides answers to aggregation queries with valid confidence intervals. We show that InQuest can achieve the same root mean squared error (RMSE) as the streaming baselines with up to 5.0x fewer samples, and it can achieve up to 1.9x lower RMSE than ABae at a fixed number of oracle invocations. We demonstrate these performance improvements on evaluation queries with and without a predicate. We perform a lesion study which shows that each component of InQuest is critical for it to achieve high performance. We further demonstrate that InQuest's improvement over baselines is not sensitive to the setting of its most significant free parameters. We analyze InQuest's cost and accuracy improvements, as well as the effect that proxy quality has on its evaluation results. Finally, we show that InQuest is resilient to sudden shifts in the stream parameters: on a set of 100 synthetic datasets that we constructed in an adversarial fashion, InQuest outperforms our streaming baselines on the RMSE metric by 1.13x-1.42x and performs within 0.99x-1.03x of ABae.

In summary, our paper makes the following contributions:

(1) We propose an algorithm for optimizing aggregation queries over multiple modalities of unstructured streams of data.
(2) We analyze the algorithm and show that its sample allocation and expected error converge at quantitative rates under certain assumptions.
(3) We evaluate the algorithm on a set of real-world video and text datasets and demonstrate significant improvement over baselines on the RMSE metric.
(4) We demonstrate that even when our theoretical assumptions do not hold, InQuest empirically outperforms our streaming baselines and is competitive with a state-of-the-art batch setting algorithm.

2 OVERVIEW AND QUERY SEMANTICS

We present an overview of the queries that InQuest optimizes. We first describe our target problem setting and specify our query syntax and semantics. We then provide example queries before defining formal notation for our problem setting.

2.1 Overview

Target setting. InQuest targets streaming aggregation queries with or without a predicate. We assume the query's statistic of interest and predicate (if present) can be computed directly by the oracle or easily derived from its output(s). InQuest supports streaming queries using the `AVG`, `SUM`, or `COUNT` aggregations.

Proxies. We assume the user provides a proxy model which returns a real number in some bounded range (e.g., [0, 1]). InQuest makes no assumptions about proxy quality, but proxies that are more correlated with the target statistic will generate more accurate query results. These proxies can be orders of magnitude cheaper to execute than the oracle (e.g., over 4,000 frames-per-second (fps) for the proxy compared to 3 fps for the oracle [30]). Thus, we make the standard assumption that proxies can be computed in an online fashion over the entire stream without buffering [11, 24].

2.2 Query Syntax and Semantics

We show the query syntax for InQuest in Figure 2. We model our syntax after the Apache Flink SQL language with some minor extensions [31]. Similar to unstructured AQP systems [24, 25, 27], a user provides InQuest with a sampling budget, a proxy model,

```
SELECT { AVG | SUM | COUNT } ({field | EXPR(field)})
FROM streaming_dataset
[WHERE filter_predicate]
TUMBLE(column, interval)   ▷ Tumbling window to define segment length
ORACLE LIMIT o                    ▷ oracle invocations per segment
[DURATION interval]           ▷ duration for non-continuous queries
USING proxy
```

Figure 2: Syntax for InQuest which is based on Apache Flink SQL syntax. Users provide a statistic to compute, a dataset, a segment length defined by a tumbling window, an oracle limit per segment, and a proxy model for computing proxy scores in real-time. Users may optionally provide a predicate and/or a query duration (for non-continuous queries).

and an oracle. The user additionally may specify a statistic (i.e., an expression) to compute on each record and an aggregation function (one of `AVG`, `SUM`, or `COUNT`). We assume that any statistic provided by the user is cheap to compute given the output of the oracle. InQuest also requires the user to specify a tumbling window [16], whose `interval` defines the length of each segment, along with a budget of oracle invocations per segment. The `column` for the tumbling window may be a time-based column or a column specifying each record's index in the stream. The `interval` can similarly be a time-based range (e.g., `INTERVAL '1' HOUR`) or it can specify a number of stream records (e.g., `INTERVAL 10,000 FRAMES`). Finally, InQuest extends the Apache Flink SQL syntax by allowing users to specify a `DURATION` for non-continuous queries [6].

Given these inputs, InQuest computes an approximate answer to the query. InQuest aims to provide answers that minimize the mean squared error (MSE) between the approximate result and the ground-truth query result. While higher quality proxies will lead to more accurate query answers, InQuest will produce an estimate regardless of proxy quality. Query answers can be provided in real-time for both continuous and non-continuous queries, although non-continuous queries will have a final answer provided at the end of the specified `DURATION`.

2.3 Examples

Traffic analysis. Consider an urban planner that would like to monitor traffic at an intersection in real-time. The planner wishes to know the per-frame average number of cars that pass through an intersection. The planner could submit the following continuous query to InQuest:

```
SELECT AVG(count(car)) FROM video
TUMBLE(frame_idx, INTERVAL '108,000' FRAMES)
ORACLE LIMIT 1,000
USING proxy_count_cars(frame)
```

where `count(car)` is computed using an objection detection DNN and `proxy_count_cars` could be computed via an embedding index for unstructured data [29]. In this setting, `proxy_count_cars` returns an estimate of the car count for every frame. The user specifies that each segment should span 108,000 frames (i.e., one hour at 30 fps) and receive a budget of 1,000 oracle invocations.

Twitter Sentiment. Consider a journalist that is interested in understanding public sentiment during a presidential debate. For example, the journalist may wish to compute the total number of tweets with positive sentiment that mention a specific candidate. The journalist can submit the following query:

```
SELECT COUNT(positive(tweet)) FROM twitter
TUMBLE(tweet_timestamp, INTERVAL '30' MINUTES)
WHERE mentions_candidate(tweet)
ORACLE LIMIT 5,000
DURATION INTERVAL '4' HOURS
USING proxy_mentions_candidate_pos(tweet)
```

The `mentions_candidate` predicate is used to filter for tweets that mention the candidate of interest. A large NLP model such as BERT [14] could be used to compute the predicate and the sentiment of the tweet. The proxy could be computed using a smaller NLP model (e.g., fasttext [8]) which would generate a probability in [0, 1] that the tweet mentions the candidate in a positive manner. Since the presidential debate (and post-debate analysis) will only last approximately 4 hours, the user also specifies a `DURATION`.

2.4 Query Formalism

Formally, let $\mathcal{D} = \{x_i\}$ be a streaming dataset of records and let $O(x_i)$ be the oracle predicate. By definition, $O(x_i) \in \{0, 1\}$ in the predicate case and $O(x_i) = 1, \forall x_i \in \mathcal{D}$ in the case without a predicate. We define $\mathcal{D}^+ = \{x \in \mathcal{D} : O(x) = 1\}$ to be the subset of the stream that satisfies the query predicate. We further define $X_i = f(x_i) \in \mathbb{R}$ to be the expression the query aggregates over, N to be the per-segment sampling budget, and T to be the number of processed segments.

InQuest computes $\mu = \sum_{x \in \mathcal{D}^+} f(x)/|\mathcal{D}^+|$ via an approximation $\hat{\mu}$, with its total sampling budget NT up to the current segment. We measure query result quality by the MSE, i.e., $|\mu - \hat{\mu}|^2$.

3 INQUEST DESCRIPTION AND QUERY PROCESSING

We describe InQuest for accelerating aggregation queries on streams of unstructured data. We first provide intuition for InQuest's design and define relevant sampling terminology. We then discuss the challenges of the problem setting before providing an overview of how InQuest addresses these challenges. Finally, we provide the pseudocode for InQuest and its subroutines. Formal notation used throughout this section is presented in Table 1.

3.1 Background and Algorithm Intuition

Our first design decision for InQuest was to leverage *stratified sampling* [38] to compute precise query estimates with standard frequentist bounds. Stratified sampling is a method in which the target population (i.e., the stream) is divided into distinct sub-populations (i.e., strata) which are sampled from independently. The objective is to stratify the target population such that elements in each stratum are similar to one another in terms of a statistic of interest. An estimate over the entire population can then be computed with smaller error (relative to uniform sampling) by aggregating lower variance estimates from each stratum.

In order to perform stratified sampling over streams, we must first determine the boundaries of our strata (in terms of proxy scores). As illustrated in the first step of Figure 3, InQuest performs *pilot sampling* [10] to accomplish this task. Pilot sampling is a

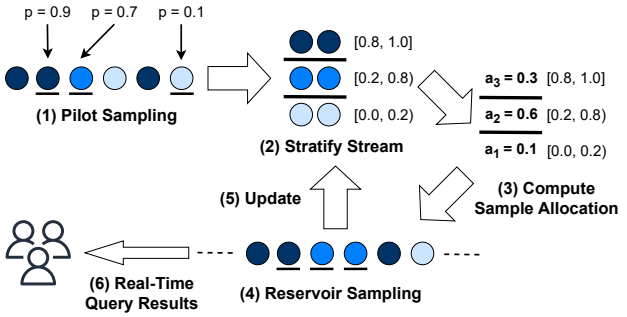

Figure 3: InQuest's high-level workflow. By design, InQuest uses stratified sampling to reduce the variance of its query estimates. It uses pilot sampling to compute an initial stratification and then updates it using the history of oracle samples. Users can extract query estimates in real-time.

technique in which a fraction of one's sampling budget is used to produce an initial estimate of some quantity (in our case, the ideal strata boundaries).

Once InQuest has constructed initial strata (Figure 3, step 2) it needs to allocate its sampling budget efficiently across the strata. There is a known optimal allocation for stratified sampling (see Section 4.2), but computing it requires perfect knowledge of quantities such as the strata standard deviations. In light of this, InQuest approximates the optimal allocation using the oracle predictions from its predicate matching pilot samples (Figure 3, step 3).

Estimating the optimal allocation using pilot samples comes with risk, because variance in the pilot sample may result in sampling few (or potentially 0) predicate matching samples in one or more strata. This outcome is considerably more likely in the stream setting, where the number of predicate matching samples can fluctuate as a function of time. If no predicate matching samples are drawn in a stratum, the optimal allocation would assign 0 samples to that stratum, thus leading to catastrophic under-allocation. InQuest uses *defensive sampling* [37] to protect against this outcome. In this context, defensive sampling is the practice of allocating a fraction of one's sampling budget evenly across all strata to ensure a minimum number of samples is allocated to each stratum (Figure 3, step 3).

Once its per-stratum sampling budget is allocated, InQuest must finally determine which samples to draw from the stream. This is non-trivial in the stream setting, because InQuest cannot know ahead of time which records will fall in each stratum. A naive solution to this problem would be to greedily sample records that fall in each stratum until the sampling budget is exhausted. However, this would produce a sample that is biased towards the beginning of the stream. To overcome this issue, InQuest makes use of *reservoir sampling* [1, 2]. Reservoir sampling is a technique that is guaranteed to sample stream records uniformly in time, without prior knowledge of the stream length. This enables InQuest to produce unbiased samples for each stratum in the stream (Figure 3, step 4).

Stream parameters, such as the strata standard deviations, can shift over time. Thus, InQuest processes the stream in segments and updates its stratification and sample allocation at the end of each segment (Figure 3, step 5). Finally, a user may retrieve InQuest's latest query estimate at any point in time (Figure 3, step 6).

Table 1: Summary of notation.

Symbol	Description
$\mathcal{D}$	Streaming dataset of records
$\mathcal{S}$	Stratification, i.e., k strata
$\mathcal{P}(x)$	Proxy model
T	Number of segments (including pilot segment)
N	Per-segment user-specified sampling budget
N_1	Per-segment defensive sample budget
N_2	Per-segment dynamic sample budget
K	Number of strata
$O(x)$	Oracle predicate
$\mathcal{D}_{tk}$	Set of dataset records in segment T and stratum k
$X_{tk,i}$	ith sample from $\mathcal{D}_{tk}$
X_{tk}	Set of samples drawn from $\mathcal{D}_{tk}$
X^+_{tk}	Set of predicate matching samples drawn from $\mathcal{D}_{tk}$
p_{tk}	Predicate positive rate
w_{tk}	$\lvert D_{tk}\rvert p_{tk} / \sum \lvert D_{tj}\rvert p_{tj}$
σ_{tk}	True std. dev. of the samples in $\mathcal{D}_{tk}$
a^*_{tk}	Optimal fraction of N_2 allocated to $\mathcal{D}_{tk}$
$f(x)$	Statistic function

3.2 InQuest Algorithm

Challenges. Our problem setting involves a number of key challenges. Similar to some prior work [27], we do not know the correlation between the proxy model $\mathcal{P}(x)$ and the ground-truth statistic function $f(x)$ ahead of time. We also do not have prior knowledge of the quantities σ_{tk} and p_{tk}. This prevents us from using standard AQP techniques [9, 39] to leverage this information and pre-compute an optimal allocation of our sampling budget.

The streaming nature of our problem creates additional challenges. In particular, the distributions of $\mathcal{P}(x)$ and $f(x)$, and the related quantities σ_{tk} and p_{tk}, are also a function of time. Thus, even if we calibrate our sample allocation based on the history of these distributions, we have no guarantees that these distributions will not change in the future.

Overview. The challenges highlighted above present unique difficulties for InQuest. The optimal allocation of our sampling budget N to the strata $\mathcal{D}_{tk}$ depends on the per-strata standard deviations of the ground-truth statistic and predicate positivity rates [36]. Without prior knowledge of σ_{tk} and p_{tk} we cannot directly compute the optimal stratified sampling allocation a^*_{tk}. Instead, we must estimate σ_{tk}, p_{tk}, and a^*_{tk} using previously drawn samples. Furthermore, the standard deviations σ_{tk} and predicate positive rates p_{tk} can vary from segment-to-segment. This means that our estimates of σ_{tk}, p_{tk}, and a^*_{tk} are susceptible to distribution shifts in the stream of data. Finally, in standard stratified sampling it is often beneficial to stratify the dataset such that each stratum contains a roughly equal number of records. Since we do not know the distribution of proxy values ahead of time, it is difficult (without buffering the entire stream) for us to construct our stratification $\mathcal{S}_{tk}$ such that each stratum will contain an equal number of records.

Algorithm 1 Pseudocode for InQuest. InQuest performs pilot sampling to initially stratify the dataset. It then performs stratified reservoir sampling on each segment as it iteratively updates $\hat{a}_{tk}$

function InQuestPilot($\mathcal{D}_1$, N_{pilot}, K)
  $X_1 \leftarrow \text{UniformSampling}(D_1, N_{\text{pilot}})$
  $X_1^+ \leftarrow \{x | x \in X_1, O(x) = 1\}$
  return X_1, X_1^+

function InQuest($\mathcal{D}$, O, $\mathcal{P}$, K, N_1, N_2)
  $X_1, X_1^+ \leftarrow \text{InQuestPilot}(\mathcal{D}_1, N_1 + N_2, K)$
  for $t \in [2, 3, \dots]$ **do**
    $\hat{\mathcal{S}}_t \leftarrow \text{GetStrata}(\mathcal{P}, \mathcal{D}_{t-1}, K, \alpha, \mathcal{S}_{<t})$
    $\hat{a}_t \leftarrow \text{GetAlloc}(\mathcal{D}_{t-1}, K, N_1, N_2, a_{<t}, X_{t-1}, X_{t-1}^+)$
    $\mathcal{D}_{t1}, \dots, \mathcal{D}_{tK} \leftarrow \text{SplitStream}(\mathcal{D}_t, \hat{\mathcal{S}}_t)$
    for $k \in [1, \dots, K]$ **do**
      $X_t \leftarrow \text{ReservoirSampling}(\mathcal{D}_{tk}, N\hat{a}_{tk})$
      $X_t^+ \leftarrow \{x | x \in X_t, O(x) = 1\}$
  $\hat{\mu} \leftarrow \text{GetPrediction}(X_1, \dots, X_T, X_1^+, \dots, X_T^+, \mathcal{D})$
  return $\hat{\mu}$

To address this, InQuest trades off learning from the history of the stream with the need to adapt to shifts in the distribution of proxy values, σ_{tk}, and p_{tk} across segments. InQuest does this by updating its stratification and sample allocation under a weighted moving average. Additionally, InQuest reserves defensive samples to increase its resilience to extreme shifts in the distributions of $\mathcal{P}(x)$ and $f(x)$. On some datasets we found that InQuest could suffer catastrophic failures without defensive sampling. Specifically, if the sample standard deviation $\hat{\sigma}_{tk}$ was close to 0 (or if no predicate matching records were sampled in the predicate setting), InQuest could undersample a stratum for the remainder of the query.

Formal description. Recall our notation from Table 1. In particular, note that $O(x)$ is the oracle predicate, $P(x)$ is the proxy predicate, and $\mathcal{D}$ is our streaming dataset of records. We define $\mathcal{D}_{tk}$ to be the subset of records in segment t and stratum k. Finally, we denote $X_{tk,i}$ and $X_{tk,i}^+$ to be the ith sample and the ith predicate matching sample drawn from $\mathcal{D}_{tk}$, respectively.

The free parameters of InQuest include its sampling budgets (N_1 and N_2), the number of strata (K), and the smoothing parameter for its weighted moving averages (α). InQuest will also compute several other quantities, including sample means, predicate positive rates, and allocations ($\hat{\mu}_{tk}$, $\hat{p}_{tk}$, and $\hat{a}_{tk}$).

InQuest uniformly samples N samples from the pilot fraction of the query. It then processes each query segment by first updating its stratification and sample allocation before performing reservoir sampling. We present the pseudocode for InQuest in the predicate setting in Algorithm 1. The pseudocode for the no predicate setting can be recovered by setting $X_{tk} = X_{tk}^+$ and $p_{tk} = 1$ for all t and k.

InQuest computes its stratification $\hat{\mathcal{S}}_t$ in the `GetStrata` subroutine in Algorithm 2. InQuest first stratifies the previous segment's samples by proxy value quantile, such that $1/K$ records in the previous segment fall in each strata. It then updates $\hat{\mathcal{S}}_t$ to be the exponential weighted moving average of the history of $\mathcal{S}_1, \dots, \mathcal{S}_{t-1}$.

Algorithm 2 Subroutines for InQuest.

function GetStrata($\mathcal{P}$, $\mathcal{D}_{t-1}$, K, α, $\mathcal{S}_{<t}$)
  $\mathcal{S}_{t-1} \leftarrow \text{StratifyByQuantile}(\mathcal{P}(\mathcal{D}_{t-1}), K)$
  $\hat{\mathcal{S}}_t \leftarrow \text{EWMA}(\{\mathcal{S}_1, \dots, \mathcal{S}_{t-1}\}, \alpha)$
  return $\hat{\mathcal{S}}_t$

function GetAlloc($\mathcal{D}_{t-1}$, K, N_1, N_2, $a_{<t}$, X_{t-1}, X_{t-1}^+)
  for $k \in [1, \dots, K]$ **do**
    $\hat{p}_{t-1,k} \leftarrow \frac{|X_{t-1,k}^+|}{|X_{t-1,k}|}$
    $\hat{\mu}_{t-1,k} \leftarrow \frac{\sum_{x \in X_{t-1,k}^+} f(x)}{|X_{t-1,k}^+|}$ if $|X_{t-1,k}^+| > 0$ else 0
    $\hat{\sigma}^2_{t-1,k} \leftarrow \frac{\sum_{x \in X_{t-1,k}^+} (f(x) - \mu_{t-1,k})^2}{(|X_{t-1,k}^+| - 1)}$ if $|X_{t-1,k}^+| > 1$ else 0
    $\hat{w}_{t-1,k} \leftarrow \sqrt{\hat{p}_{t-1,k}} \cdot \frac{|\mathcal{D}_{t-1,k}|}{|\mathcal{D}_{t-1}|}$
  for $k \in [1, \dots, K]$ **do**
    $a_{t-1,k} \leftarrow \frac{\hat{w}_{t-1,k}\hat{\sigma}_{t-1,k}}{\sum_{j=1}^K \hat{w}_{t-1,j}\hat{\sigma}_{t-1,j}}$
  $\hat{a}_t \leftarrow \text{EWMA}(\{a_1, \dots, a_{t-1}\}, \alpha)$
  for $k \in [1, \dots, K]$ **do**
    $\hat{a}_{tk} \leftarrow \frac{N_1/K + N_2 \hat{a}_{tk}}{N}$
  return $\hat{a}_t$

function GetPrediction($X_1, \dots, X_T, X_1^+, \dots, X_T^+, \mathcal{D}$)
  for $t \in [1, T]$ **do**
    for $k \in [1, K]$ **do**
      $\hat{p}_{tk} \leftarrow \frac{|X_{tk}^+|}{|X_{tk}|}$
      $\hat{\mu}_{tk} \leftarrow \frac{\sum_{x \in X_{tk}^+} f(x)}{|X_{tk}^+|}$
  $\hat{\mu} \leftarrow \sum_{t=1}^T \sum_{k=1}^K \hat{\mu}_{tk} \cdot \frac{\hat{p}_{tk}|\mathcal{D}_{tk}|}{\sum_{t=1}^T \sum_{j=1}^K \hat{p}_{tj}|\mathcal{D}_{tj}|}$
  return $\hat{\mu}$

The aggressiveness of the weighted moving average is controlled by the smoothing parameter α.

Next, InQuest computes its sample allocation $\hat{a}_t$ in the `GetAlloc` subroutine in Algorithm 2. InQuest first computes the previous segment's sample standard deviations $\hat{\sigma}_{t-1,k}$ and predicate positive rates $\hat{p}_{t-1,k}$. It then computes the optimal allocation a_{t-1} as a weighted average of the stratum standard deviations. InQuest then computes the sample allocation $\hat{a}_t$ to be the exponential weighted moving average of the history of $a_1, \dots, a_{t-1}$. Finally, it adjusts $\hat{a}_t$ to include N_1/K defensive samples per stratum.

Once the stratification and sample allocation are computed, InQuest performs reservoir sampling in each $\mathcal{D}_{tk}$. InQuest repeats this process for each segment before finally computing its prediction $\hat{\mu}$ as a weighted average of the sample means $\hat{\mu}_{tk}$ in the `GetPrediction` subroutine in Algorithm 2.

Setting parameters. By default, InQuest uses the parameter settings: $K = 3$, $\alpha = 0.8$, and $N_1 = 0.1$. In Section 5 we demonstrate that these parameters achieve strong performance results on six real-world and 100 synthetic datasets, relative to both streaming and batch setting algorithms. Advanced users may optionally tune these parameters to optimize performance on their own datasets.

As a general guideline, we recommend setting N_1 to be small (~ 5-10% of N) and setting N and K such that one would reasonably expect to get at least 20-100 samples per segment and stratum.

Confidence interval. We use the bootstrap to compute CIs. One method for showing that the bootstrap is valid is to demonstrate its asymptotic validity. The asymptotic validity of sampling with stochastic draws follows from the analysis in [28]. We can also use a standard subgaussian tail bound, but they give similar results.

4 THEORETICAL ANALYSIS

We analyze InQuest from a theoretical perspective and show that its allocation and expected error converge at quantitative rates for stationary streams. We first show that InQuest's sample allocation converges to the optimal stratified sampling allocation at a rate $O\Big(\frac{1}{N_1(t-1)}\Big)$. We then show that InQuest's expected MSE converges to zero at a rate $O\Big(\frac{1}{N_1} + \frac{N_1}{N_2^2} + \frac{1}{N_2\sqrt{N_1}} + \frac{1}{N_2\sqrt{N_1 t}}\Big)$.

4.1 Notation and Preliminaries

Notation. Recall our notation in Table 1. We denote p_{tk} and σ_{tk} to be the predicate positive rate and standard deviation of $\mathcal{D}_{tk}$ respectively. We further define μ_t to be the segment mean, w_{tk} to be the fraction of dataset records that fall in $\mathcal{D}_{tk}$, and a_{tk} to be the allocation of our sampling budget N.

Assumptions. For subsections 4.3 and 4.4, we assume that our streaming dataset follows a stationary distribution. Specifically, we assume that:

$$\sigma_{tk} = \sigma_{rk} : \forall t, r \in T \tag{1}$$

$$p_{tk} = p_{rk} : \forall t, r \in T \tag{2}$$

$$w_{tk} = w_{rk} : \forall t, r \in T \tag{3}$$

While these assumptions may not hold true for real-world datasets, they are important for making our proofs tractable. In Sections 5.2 and 5.6 we present empirical evidence that InQuest performs well relative to baselines even when these assumptions break down.

Recall that $X_{tk,i}$ is the ith predicate matching sample drawn from $\mathcal{D}_{tk}$. We assume that $X_{tk,i}$ is a sub-Gaussian random variable with nonzero standard deviation. This enables us to upper bound functions that sum sub-Gaussian variables (e.g., μ_{tk} and σ^2_{tk}) with constants such as $C^{\mu_{tk}}$ and $C^{\sigma^2_{tk}}$. We further assume that at least one stratum has non-zero p_{tk}.

4.2 Optimal Stratified Sampling Allocation with Perfect Information

We begin by analyzing the optimal allocation of our dynamic sample budget N_2 in segment t. We assume perfect knowledge of σ_{tk} and p_{tk} and that we deterministically draw $|X^+_{tk}| = p_{tk}\big(\frac{N_1}{K} + N_2 a_{tk}\big)$ samples from each stratum. We present the analysis for the setting with a predicate, but note that these results also hold for the no predicate setting where $p_{tk} = p_t = 1$.

Proposition 1. *Assume that σ_{tk} is known and we draw $|X^+_{tk}| = p_{tk}\big(\frac{N_1}{K} + N_2 a_{tk}\big)$ samples per stratum in segment $t > 2$ (up to rounding effects). Then the choice $a_{tk} = a^*_{tk}$ that minimizes the MSE of the unbiased estimator $\hat{\mu}_t = \sum_{k=1}^K w_{tk} \cdot \frac{\sum_{x \in X^+_{tk}} f(x)}{|X^+_{tk}|}$ is:*

$$a^*_{tk} = \frac{|\mathcal{D}_{tk}|\sqrt{p_{tk}}\sigma_{tk}}{\frac{N_2}{N}\sum_{j=1}^K |\mathcal{D}_{tj}|\sqrt{p_{tj}}\sigma_{tj}} - \frac{N_1}{N_2 K} \tag{4}$$

Proposition 2. *Suppose the conditions in Proposition 1 hold. Then the expected MSE of the estimator $\hat{\mu}_t$ under the allocation a^*_{tk} is*

$$\mathbb{E}[(\hat{\mu}^*_t - \mu_t)^2] = \sum_{k=1}^K \frac{w^2_{tk}\sigma^2_{tk}}{p_{tk}\big(\frac{N_1}{K} + N_2 a^*_{tk}\big)} \tag{5}$$

$$= \frac{1}{N p^2_{all}} \sum_{k=1}^K |\mathcal{D}_{tk}|\sqrt{p_{tk}}\sigma_{tk}\left(\sum_{j=1}^K |\mathcal{D}_{tj}|\sqrt{p_{tj}}\sigma_{tj}\right) \tag{6}$$

Where p_{all} is defined as

$$p_{all} = \sum_{j=1}^K |\mathcal{D}_{tj}| p_{tj} \tag{7}$$

Our expression for a^*_{tk} shows that the optimal allocation is weighted towards strata with larger $|\mathcal{D}_{tk}|$, p_{tk}, and σ_{tk}. Intuitively, we want to spend our sampling budget on strata that are more likely to contain predicate matching records. Furthermore, strata with greater σ_{tk} will generally require more samples to get an accurate estimate of μ_{tk}.

The expression for the expected error shows that larger strata standard deviations will lead to an increase in the error. The expected error also increases inversely with respect to p^2_{all}, where p_{all} is a weighted sum of the strata predicate positive rates. Intuitively, as p_{all} goes to 0 it becomes harder for InQuest to compute accurate estimates of μ_{tk} because it becomes increasingly unlikely that InQuest will draw a sample in X_{tk} that matches the predicate. Finally, the expected error decreases linearly with respect to our sampling budget N.

4.3 InQuest Sample Allocation Converges to Optimal Stratified Sampling Allocation

We analyze InQuest's dynamic sample allocation and prove that it converges to the optimal allocation at a quantitative rate under the assumptions stated in subsection 4.1. For this analysis we further assume that $\alpha = 0$, i.e., we compute the update to the sample allocation $\hat{a}_{tk}$ based on the unweighted history of the samples. We provide the theorem statement but defer the full proof to an extended technical report [42].

Theorem 1. *Under the assumptions stated in subsection 4.1 and with high probability over the samples drawn in segments $[2, \ldots, t]$*

$$\mathbb{E}[(\hat{a}_{tk} - a^*_{tk})^2] \leq O\Big(\frac{1}{N_1(t-1)}\Big) \tag{8}$$

Equation 8 shows that InQuest's sample allocation converges to the optimal allocation at a rate that decreases linearly as a function of N_1 and t. The product $N_1(t-1)$ represents the total number of defensive samples in all segments leading up to segment t.

4.4 InQuest Error Convergence

We analyze InQuest's expected error and prove that it converges at a quantitative rate under the assumptions stated in subsection 4.1. For this analysis we further assume that $\alpha = 0$. We provide the theorem statement, but once again defer the full proof to an extended technical report [42].

Theorem 2. *Under the assumptions stated in subsection 4.1 and with high probability over the samples drawn in segments* $[1, \ldots, t]$

$$\mathbb{E}[(\hat{\mu}_t - \mu_t)^2] \leq O\Big(\frac{1}{N_1} + \frac{N_1}{N_2^2} + \frac{1}{N_2\sqrt{N_1}} + \frac{1}{N_2\sqrt{N_1 t}}\Big) \tag{9}$$

Furthermore, if $N_1 = N_2$ *then this simplifies to*

$$\mathbb{E}[(\hat{\mu}_t - \mu_t)^2] \leq O\Big(\frac{1}{N}\Big) \tag{10}$$

4.5 Understanding InQuest

We provide proof sketches for the theorems and discuss some aspects of the analysis of InQuest that are of broader interest.

4.3.1 Proof Sketch: InQuest Allocation Convergence. We use concentration inequalities to bound our random variables, specifically $p_{<tk}$ and $\sigma_{<tk}$. We then compute an upper bound on the expected mean squared error of the difference between $\hat{a}_{tk}$ and a^*_{tk}. We separately compute the upper bound for cases where $p_{<tk}$ is small (i.e., less than $\frac{1}{N_1}$) and cases where $p_{<tk}$ is large. We simplify the expectation and conclude that our allocation error converges at a rate of $O\Big(\frac{1}{N_1(t-1)}\Big)$.

4.4.1 Proof Sketch: InQuest Error Convergence. We use concentration inequalities to bound our random variables, including p_{tk} and σ_{tk}, as well as on other quantities derived from these variables. We use these bounds to compute a high probability lower bound on the number of predicate matching samples drawn for each stratum (i.e., $|X^+_{tk}|$) for the case where p_{tk} is large (i.e., larger than $\frac{1}{N_1}$). We then derive the upper bound on the expected error for strata where p_{tk} is large and show that the error for the remaining strata becomes negligible. Finally, we simplify our expectation and conclude that our error converges at a rate of $O\Big(\frac{1}{N}\Big)$.

4.4.2 Challenges. We discuss several challenges in the analysis of InQuest. Recent work has analyzed using stochastic draws in the batch setting, where the dataset can be stratified perfectly and pilot sampling can be performed with samples drawn from the entire dataset [27]. We extend this work using stochastic draws to the stream setting and show that InQuest can achieve optimal performance on stationary datasets.

Estimating key quantities. Prior work in stratified sampling assumes that features of the data distribution in each stratum, such as p_{tk} and σ_{tk}, are known [36]. It then uses this knowledge to construct optimal sample allocations. In contrast, InQuest has no prior knowledge of these quantities and must estimate them from samples it draws stochastically. For values of p_{tk} that are small relative to our sample budget N (e.g., $p_{tk} < \frac{1}{N_2}$), InQuest may not draw a single predicate matching sample, thus making it impossible to accurately estimate p_{tk} and σ_{tk}.

Stochastic sample sizes. In the predicate setting InQuest may sample records that do not satisfy the predicate. As a result, the number of predicate matching samples in each $\mathcal{D}_{tk}$ is stochastic. This is in contrast to standard stratified sampling, which assumes a deterministic number of draws from each stratum. In the case where both p_{tk} and $|X_{tk}|$ are large, InQuest will draw approximately $p_{tk}|X_{tk}|$ samples which will result in estimates with similar quality to an estimator with $p_{tk}|X_{tk}|$ deterministic samples. However, for small p_{tk} this no longer holds true.

Recursive Definitions. By design, InQuest samples the stream and updates its allocation at discrete intervals throughout the query. The allocation $\hat{a}_{tk}$ depends on the history of samples drawn in segments $[1, \ldots, t-1]$. Specifically, it's a function of $\hat{p}_{tk}$ and $\hat{\sigma}_{tk}$. In turn, $\hat{p}_{tk}$ and $\hat{\sigma}_{tk}$ depend on the number of samples drawn from $\mathcal{D}_{tk}$. This means they are a function of the allocation $\hat{a}_{tk}$. The recursive nature of these definitions makes it challenging to apply meaningful concentration inequalities on these random variables.

5 EVALUATION

We evaluated our algorithm on six real-world video and text datasets. We first describe our experimental setup and baselines. We then show that InQuest outperforms the stream setting baselines on all datasets we consider, achieving the same root mean squared error (RMSE) with up to 5.0x fewer samples. We further demonstrate that InQuest outperforms ABae [27]—a state-of-the-art algorithm for the batch setting—by up to 1.9x on the RMSE metric at a fixed sample budget. We then show that each of InQuest's major components contributes to its performance and that it is not sensitive to the setting of its parameters. Finally, we analyze InQuest's cost and accuracy improvements, examine effect that proxy quality has on its evaluation results, and demonstrate that InQuest is resilient to rapid changes in the stream parameters.

5.1 Experimental Setup

Datasets, proxies, and oracles. We considered six real-world video and text datasets (Table 2). The video datasets are commonly used for video analytics evaluation [24, 25, 27, 32]. The text dataset is publicly available on Kaggle and contains 3M+ tweets between users and customer support Twitter accounts [23]. For each video dataset we generated proxy scores from TASTI embeddings that we created with a pre-trained ResNet-18 model [19, 29]. Our oracle labels were computed using a Mask R-CNN model [18]. For our text dataset we generated proxy scores using a fasttext model, while our oracle labels were computed using a HuggingFace BERT model trained on English language tweets [22, 33].

Evaluation queries. We evaluated the baselines and InQuest on each dataset using two queries, one with a predicate and one without. The queries with a predicate were all of the form:

```
SELECT AVG(expr(record)) FROM dataset
WHERE filter_predicate
TUMBLE(record_idx, INTERVAL '100,000' RECORDS)
ORACLE LIMIT N
DURATION INTERVAL '500,000' RECORDS
USING proxy
```

For our video datasets, `expr` was `count_boats` for the `grand-canal` and `rialto` datasets and `count_cars` for the rest. The predicate

Table 2: Summary of datasets, predicates, predicate positivity rates p, and proxy correlation to the groundtruth statistic r (Pearson product-moment correlation coefficient).

Dataset	Predicate	p	r
archie	At least one car	0.50	0.92
customer-support	Is customer tweet	0.56	0.79
grand-canal	At least one boat	0.60	0.91
night-street	At least one car	0.37	0.92
rialto	At least one boat	0.89	0.91
taipei	At least one car	0.63	0.87

was `count_boats(record) > 0` and `count_cars(record) > 0`, respectively. For our text dataset `expr` was `sentiment` and the predicate was `is_customer_tweet(record)`. The queries without a predicate were identical to the one shown above with the `WHERE` clause removed. Each query allocated 10% of the oracle budget for defensive sampling.

Streaming methods evaluated. We compared our algorithm against two baselines for the stream setting: uniform sampling and stratified sampling with fixed strata and fixed sample allocations. For uniform sampling, we precomputed a set of N frames (where N is the oracle budget) to sample between the query submission time and the end of the query's `DURATION`. We then called the oracle on these records and computed our query estimate by averaging the per-record statistic on the sampled frames:

```
SELECT AVG(expr(record)) FROM uniform_sample(dataset)
ORACLE LIMIT N
DURATION INTERVAL '500,000' RECORDS
```

For queries with a predicate, the estimate was only computed using the statistic values from the predicate-matching samples:

```
SELECT AVG(expr(record)) FROM uniform_sample(dataset)
WHERE expr(record) > 0
ORACLE LIMIT N
DURATION INTERVAL '500,000' RECORDS
```

For the stratified sampling baseline, we executed evaluation queries similar to the ones used for InQuest. We also performed stratified sampling within each segment. However, each segment and stratum pair (i.e., $\mathcal{D}_{tk}$) received a fixed oracle budget of $\frac{N}{K}$ samples and maintained a fixed stratification of $k_1 = [0, 0.33]$, $k_2 = [0.33, 0.67]$, and $k_3 = [0.67, 1.0]$. Due to the streaming nature of our queries, we performed reservoir sampling within each $\mathcal{D}_{tk}$ to ensure that the oracle was applied uniformly at random. We then computed our estimate for each segment as a weighted average of the aggregation function `AGG` (one of `AVG`, `SUM`, or `COUNT`) applied to the samples from each $\mathcal{D}_{tk}$:

$$\hat{\mu}_t = \sum_{k=1}^{K} \hat{w}_{tk} \cdot \text{AGG}\big(\{f(x)|x \in S_{tk}\}\big)$$

Where the weight $\hat{w}_{tk}$ is the estimate of the fraction of predicate matching samples that fall in $\mathcal{D}_{tk}$:

$$\hat{p}_{tk} = \frac{|\{x|O(x) = 1, x \in S_{tk}\}|}{|S_{tk}|} \tag{11}$$

$$\hat{w}_{tk} = \frac{|\mathcal{D}_{tk}| \cdot \hat{p}_{tk}}{\sum_{i=1}^{K} |\mathcal{D}_{ti}| \cdot \hat{p}_{ti}} \tag{12}$$

For queries without a predicate $\hat{p}_{tk} = p_{tk} = 1$ which meant our estimate $\hat{w}_{tk}$ was computed exactly. In our baseline experiments for stratified sampling with fixed strata we set $K = 3$ and configured our segment length such that there were $T = 5$ segments.

Batch methods evaluated. In order to compare InQuest to prior work, we also evaluated ABae [27] using near-identical evaluation queries (minor syntax tweaks were necessary for the batch setting). We chose ABae for the comparison because it also provides approximate answers to aggregation queries with valid confidence intervals. To run the evaluation, we presented our streaming datasets to ABae as if they were batch datasets. We ran ABae with sample reuse, $K = 3$, and allocated 15% of its budget to pilot sampling.

ABae has the advantage of observing the proxy score distribution over the entire dataset, which allows it to compute an optimal stratification and sample allocation before it begins sampling. In contrast, InQuest does not have the benefit knowing the proxy score distribution prior to sampling. In spite of this, we find the comparison useful for contextualizing InQuest's results, and we show that InQuest outperforms ABae on our key metric of interest.

Metrics. Our primary metric of interest is the RMSE between each method's estimate of the expression in the `SELECT` clause and the oracle value. In particular, we measure the RMSE on each segment of the query and evaluate each method by computing the median RMSE across all query segments. We evaluated the RMSE at different oracle budgets representing 0.1 - 1% of the total records in each query. We additionally compared the number of samples needed to achieve a fixed error target.

Implementation. We implemented our algorithm, baselines, and experimental evaluation in Python. Our open-sourced code can be found at https://github.com/stanford-futuredata/InQuest.

5.2 InQuest End-to-end Performance

We first investigated whether or not InQuest outperforms our baselines on the median segment RMSE metric. For each dataset we evaluated the uniform sampling baseline, the stratified sampling baseline, ABae, and InQuest on the evaluation queries with and without a predicate. As shown in Table 2, our evaluation queries in the predicate setting cover a wide range of predicate positivity rates, with 37% to 89% of records matching the predicate. We swept the oracle budget from 500 to 5000 in increments of 500 and ran 1000 trials for each oracle budget. We executed InQuest with its default hyperparameters ($K = 3$, $\alpha = 0.8$) on all datasets.

Figure 4 shows InQuest and the baselines' performance on the RMSE metric for the evaluation queries without a predicate. InQuest outperforms streaming baselines on all sampling budgets across all datasets. InQuest achieves as much as a 3.5x improvement on RMSE over streaming baselines at a fixed oracle budget, and can achieve the same RMSE with up to 5.0x fewer samples. Figure 5 shows InQuest's results on queries with a predicate. Once again,

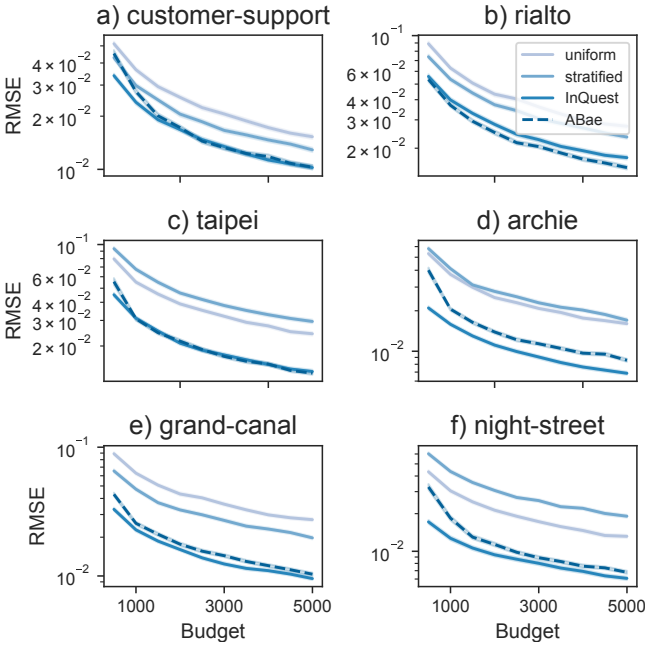

Figure 4: Sampling budget vs. median segment RMSE for baselines and InQuest on the evaluation queries with no predicate (log scale). InQuest outperforms the streaming baselines across all sampling budgets and datasets. InQuest outperforms ABae on 70.0% of oracle budgets across datasets.

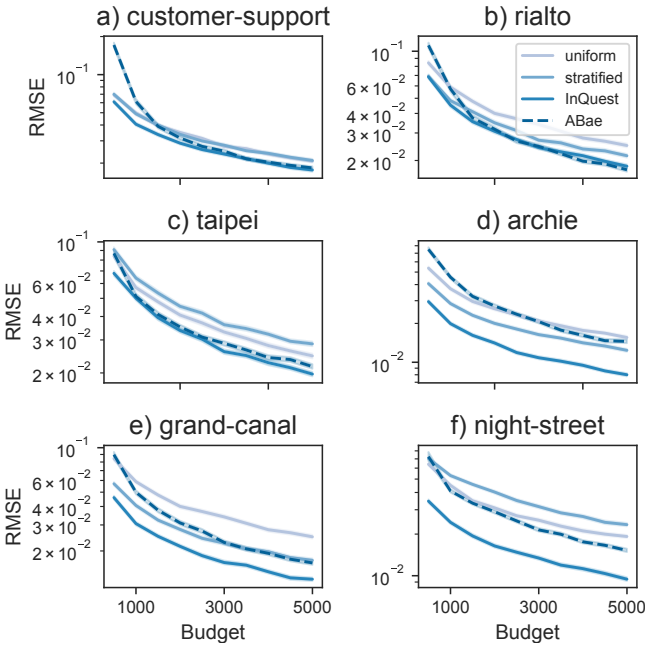

Figure 5: Sampling budget vs. median segment RMSE for baselines and InQuest on the evaluation queries with a predicate (log scale). InQuest outperforms the streaming baselines across all sampling budgets and datasets. InQuest outperforms ABae on 90.0% of oracle budgets across datasets.

Table 3: Summary of algorithm performance relative to streaming baselines and ABae in the no predicate case. RMSE errors are computed by taking the geometric mean of the average RMSE across all datasets at the specified budget.

Algorithm	$NT = 500$	$NT = 2500$	$NT = 5000$	All
$RMSE_{\text{uniform}^\dagger}$	.065	.029	.020	.030
$RMSE_{\text{stratified}^*}$	.064	.029	.020	.030
$RMSE_{\text{ABae}^\ddagger}$	.044	.015	.010	.016
$RMSE_{\text{InQuest}}$	**.032**	**.014**	**.0099**	**.015**
Improvement†	2.05x	2.03x	1.99x	2.00x
Improvement*	2.01x	2.02x	1.98x	2.00x
Improvement‡	1.40x	1.05x	1.04x	1.10x

Table 4: Summary of algorithm performance relative to streaming baselines and ABae in predicate case. RMSE errors are computed by taking the geometric mean of the average RMSE across all datasets at the specified budget.

Algorithm	$NT = 500$	$NT = 2500$	$NT = 5000$	All
$RMSE_{\text{uniform}^\dagger}$	.072	.032	.021	.033
$RMSE_{\text{stratified}^*}$	.065	.029	.020	.030
$RMSE_{\text{ABae}^\ddagger}$	.096	.027	.017	.029
$RMSE_{\text{InQuest}}$	**.049**	**.020**	**.014**	**.021**
Improvement†	1.48x	1.56x	1.58x	1.54x
Improvement*	1.32x	1.43x	1.48x	1.42x
Improvement‡	1.97x	1.32x	1.26x	1.37x

InQuest outperforms streaming baselines on all sampling budgets across all datasets. We demonstrate an improvement of up to 2.5x in RMSE at a fixed oracle budget over streaming baselines and are able to achieve the same error with up to 4.5x fewer samples.

We also compare InQuest to ABae on the median segment RMSE metric. By default, ABae only returns an estimate for the entire query. We computed per-segment estimates by selecting the subset of ABae's oracle samples within each segment. We show that InQuest outperforms ABae on 90.0% and 70.0% of oracle budgets across all datasets for queries with and without a predicate, respectively. We also compare InQuest to ABae on the RMSE metric for the full query at the end of this subsection.

Finally, we quantified InQuest's performance relative to our baselines using a single error metric aggregated across all datasets. For each dataset and algorithm, we computed the mean of the median segment RMSEs over all 1000 trials at the given oracle budget. We then computed the geometric mean of these per-dataset average RMSEs to obtain a single aggregated error metric. We present these metrics in Table 3 and Table 4. We can see that InQuest achieves the lowest error metrics, outperforming streaming baselines by a factor of 1.32x-1.58x and 1.98x-2.05x across the entire range of oracle budgets for queries with and without a predicate, respectively. Furthermore, we demonstrate that InQuest outperforms ABae by a factor of 1.04x-1.40x for queries without a predicate and by a factor of 1.26x-1.97x for queries with a predicate.

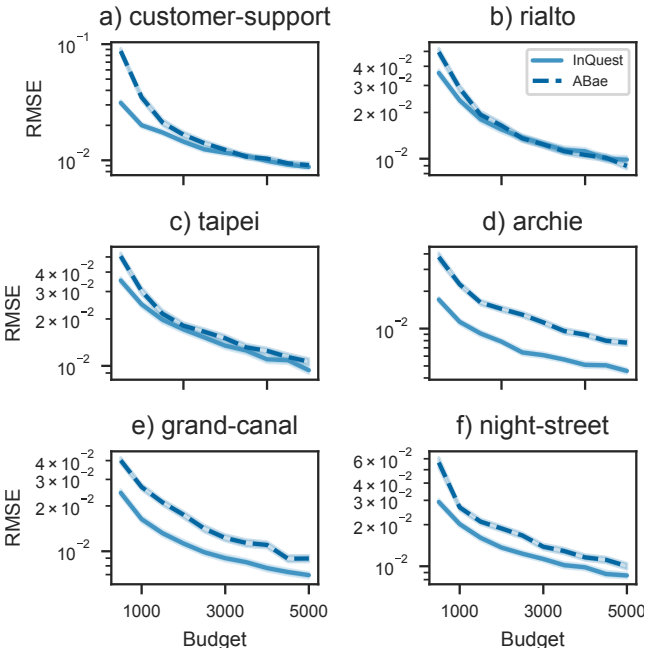

Figure 6: Sample budget vs. full query RMSE for InQuest and ABae on the evaluation queries with a predicate (log scale).

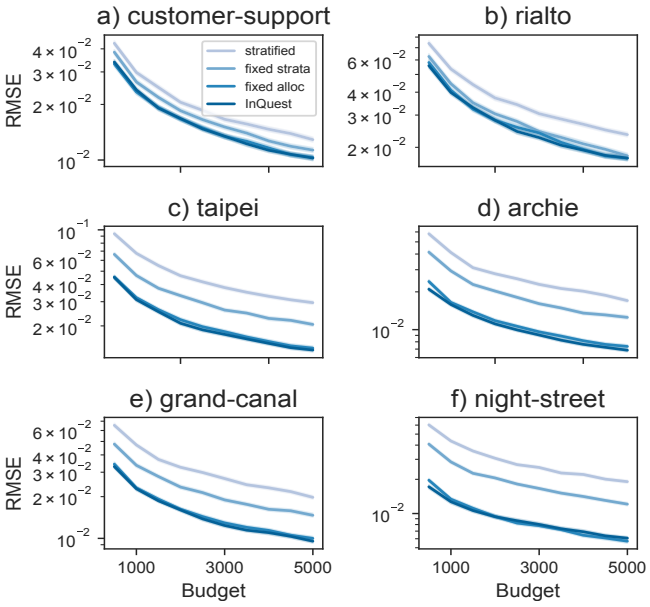

Figure 7: Lesion study in which we remove dynamic strata inference and dynamic sample allocation. As shown, both pieces of the algorithm are critical for achieving better performance across all datasets. All experiments were run on queries with no predicate.

Evaluating InQuest and ABae on Full Query. We now compare InQuest and ABae using the RMSE metric over the full evaluation query. This evaluation is useful for contextualizing InQuest's results relative to a state-of-the-art algorithm in the batch setting. We show the results for the queries with a predicate in Figure 6, and defer the figure for the queries without a predicate to an appendix [42].

When aggregating error results across all datasets we find that InQuest outperforms ABae by a factor of 1.05x-1.41x on queries without a predicate, and by a factor of 1.18-1.83x for queries with a predicate. While one might expect ABae to provide an upper bound on InQuest's performance, InQuest can benefit from its segmentation of certain streams. Specifically, if a stream is segmented over time such that $\sigma_{tk} < \sigma_k$ for enough segments $t \in [1, T]$, InQuest can achieve more accurate estimates than a batch algorithm. To summarize, InQuest can exploit the tendency in many real-world streams for proxy scores that are nearby in time to have similar values, which results in smaller σ_{tk} and ultimately smaller errors when estimating μ.

5.3 Lesion and Sensitivity Analysis

Lesion study. We investigated whether all of InQuest's components were necessary for high performance. We performed a lesion study by executing (1) InQuest, (2) InQuest with dynamic strata inference but fixed sample allocations, (3) InQuest with fixed strata but dynamic sample allocations, and (4) stratified sampling with a pilot segment. All experiments were run with 1000 trials on the evaluation queries with no predicate.

As shown in Figure 7, both dynamic strata inference and dynamic sample allocations are important for achieving high performance. In particular, removing dynamic strata inference can severely limit InQuest's ability to avoid wasting samples on strata with few predicate matching records. While removing dynamic sample allocation does not significantly affect performance on some datasets (e.g., `grand-canal`), it is necessary for achieving high performance on others (e.g., `archie`, `rialto`, and `taipei`).

Sensitivity analysis. We further investigated whether InQuest's performance was sensitive to the setting of its key parameters. Specifically, we analyzed the sensitivity of InQuest to the smoothing parameter α and to the length of its tumbling window. We ran InQuest with a budget of 5000 samples for 1000 trials while varying α and the window length. All experiments were run on the evaluation queries with no predicate.

Figure 8 shows InQuest's performance as a function of α and the window length on the `archie` dataset. InQuest's performance is relatively stable with respect to changes in α and the window length. We varied $\alpha \in [0.5, 0.9]$ in increments of 0.1 and we varied the window length such that the query contained $T \in [4, 8]$ segments. We compared InQuest to uniform sampling, which is invariant to these parameters. InQuest outperforms uniform sampling on the RMSE metric on all datasets and settings of the α parameter and the window length. We defer the plots for the other datasets to our appendix for the sake of brevity.

5.4 Cost Savings and Accuracy Improvements

We now examine how much InQuest saves on cost—both in terms of time and dollars—relative to baselines. For each algorithm we measure its accuracy using the median segment RMSE. We compute

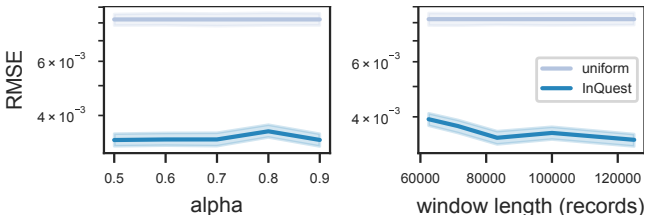

Figure 8: Sensitivity analysis of InQuest to the smoothing parameter α and our tumbling window length on the `archie` dataset. As shown above, we can see that InQuest's RMSE is fairly stable w.r.t. changes in α and the window length.

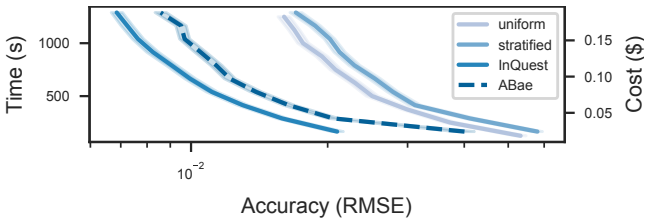

Figure 9: Time and cost in dollars as a function of accuracy for the `archie` dataset. At fixed accuracy, InQuest achieves a speed up (and cost savings) of up to 5.8x over streaming baselines and up to 1.6x over ABae.

the relative cost of running each algorithm as a function of its execution of its oracle and proxy models.

We consider the case of our video datasets, where our oracle is a Mask R-CNN model that can process 4 frames per second on an NVIDIA T4 GPU, and our proxy is a ResNet-18 model which can process 12.6k frames per second [30]. Using on-demand pricing from Amazon Web Services [15], we assume the cost of running a single NVIDIA T4 GPU on a `g4dn.xlarge` is $0.526 per hour. We compute the time (and associated cost) of running inference using our oracle and proxy models to achieve the stated accuracy. The results for the evaluation query on `archie` dataset without a predicate are presented in Figure 9. We can see that InQuest outperforms all other algorithms on the `archie` dataset in terms of accuracy at fixed cost and cost at fixed accuracy. InQuest achieves a speed up (and cost savings) of up to 5.8x over streaming baselines and up to 1.6x over ABae. InQuest outperforms the baselines on the other datasets as well, achieving worst and best-case speedups of 1.5x-8.3x over streaming baselines and 0.8x-2.0x over ABae in the no predicate case, and of 1.0x-4.1x over streaming baselines and 0.9x-2.6x over ABae in the predicate case. We omit those plots for the sake of brevity and defer them to the appendix.

5.5 Effect of Proxy Quality on Performance

In this subsection we examine how proxy score quality affects InQuest's performance on our evaluation datasets. We modified the proxy scores for our evaluation datasets by interpolating between the groundtruth statistic (i.e., perfect proxy information) and random noise. Specifically, for $\beta \in [0, 1]$ we computed:

$$\text{proxy}_i = \beta \cdot \text{g}_i + (1 - \beta) \cdot \mathbb{U}(0, 1) \tag{13}$$

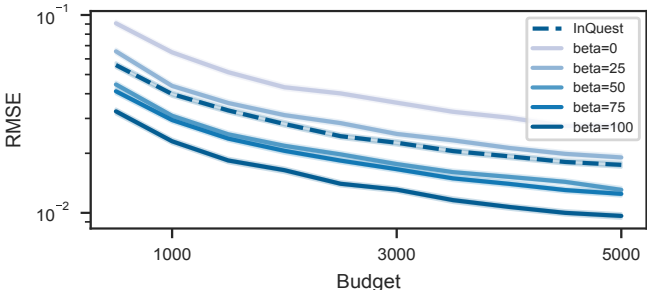

Figure 10: Proxy quality's effect on InQuest's performance on the `rialto` dataset. We plot InQuest's performance on the median segment RMSE metric as a function of β.

Where g_i is the groundtruth statistic for the i^{th} dataset record and $\mathbb{U}(0, 1)$ is a sample drawn uniformly at random from the range $[0, 1]$. After computing the proxy values for each stream we normalized them to be in the range $[0, 1]$. We constructed datasets in this fashion for $\beta \in [0.0, 0.25, 0.50, 0.75, 1.0]$. In figure Figure 10 we plot InQuest's performance on the `rialto` dataset as a function of β. We chose this dataset because its proxy's Pearson correlation coefficient is near the median for our datasets (see Table 2).

As shown in Figure 10, proxy quality can result in an orders-of-magnitude improvement of InQuest's performance. However, our current proxy scores are far from optimal, resulting in performances comparable to those with $\beta \in [0.25, 0.75]$. Thus, while we would expect ABae's performance to also improve with better proxies, we can confidently state that our performance relative to our uniform sampling baseline would greatly improve with better proxies.

5.6 Adversarial Shifts in Stream Parameters

We now investigate how one or more sudden shifts in the stream parameters p_{tk}, σ_{tk}, and μ_{tk} affects InQuest's performance.

Dataset construction. We constructed streams by randomly inserting $n = [1, 2, \ldots, 5]$ sudden shifts in the stream parameters. For each value of n we generated 20 streams, thus creating a total of 100 synthetic datasets. To generate a stream, we began by sampling n indices uniformly at random where we would suddenly shift the stream parameters. We then sampled our initial stream parameters p_{1k}, σ_{1k}, and μ_{1k} where: $p_{tk} \in [0, 1]$, $\sigma_{tk} \in [0, 3]$, and $(\mu_{t1}, \mu_{t2}, \mu_{t3}) \in ([0, 3], [3, 6], [6, 9])$.

For each value of $k \in [1, K]$ we generated a substream of samples using parameters $(p_{1k}, \sigma_{1k}, \mu_{1k})$. We then interleaved the samples from our K substreams into our final synthetic streaming dataset until we reached the sample index for a sudden shift in parameters. At every such index, we resampled p_{tk}, σ_{tk}, and μ_{tk} for all $k \in [1, K]$ and continued constructing the streaming dataset with the new stream parameters in the same fashion. Finally, we computed synthetic proxy values by interpolating the groundtruth statistic in an identical fashion to our experiments in Section 5.5. For our synthetic datasets, we used $\beta = 0.75$ to construct the proxies.

While our theoretical analysis focused on InQuest's performance on stationary streams, by construction these synthetic datasets stress test InQuest's ability to adjust to sudden changes in dynamic streams. We evaluated our streaming baselines, ABae, and InQuest

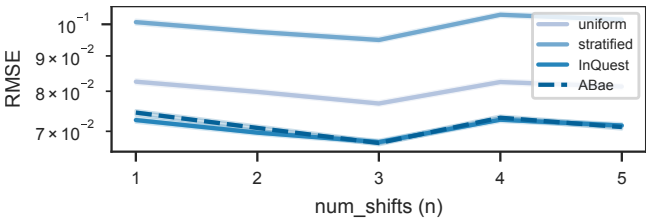

Figure 11: Analyzing the effect of shifts in the stream parameters on InQuest's performance. InQuest outperforms streaming baselines by an average of 1.13x-1.42x on the median RMSE metric when evaluated on 100 synthetic datasets.

for 1000 trials on all 100 synthetic datasets. We computed each algorithm's average median RMSE across all datasets for fixed values of n and present the results in Figure 11.

Results. InQuest consistently outperforms our streaming baselines on these synthetic datasets, and performs comparably to ABae. InQuest achieves 1.13x-1.42x improvement over our streaming baselines across the full range of $n \in [1, 5]$ distribution shifts. InQuest also performs within 0.99x-1.03x of ABae. These results provide empirical evidence that InQuest performs well even on non-stationary streams with multiple sudden shifts in the stream parameters.

6 RELATED WORK

We examine prior work in query processing as it relates to InQuest. We first discuss prior methods for processing stream queries, particularly on video datasets. We then examine other AQP systems with a focus on DNN-based queries. Finally, we discuss literature related to proxy models and their use in query processing.

Stream queries. Prior work has focused on building systems to answer queries over (semi-)structured streams of data. Similar to InQuest, these systems support continuous queries for real-time analytics, in which a user can submit a query for an indefinite period of time to compute a statistic of interest (e.g., the number of cars that pass over a sensor on the highway) [5, 12, 31, 34, 46]. However, because these systems are designed to work with structured data they do not address the use case where a DNN is needed to extract statistics of interest from the raw data stream.

Recent work has focused on answering spatiotemporal queries over streaming video, including aggregation queries [32], queries related to object co-occurrences [13], and queries related to object interactions [45]. Similar to InQuest, these systems use cascades of filters to limit the execution of an oracle to a subset of the stream. However, these systems bake their filters into various layers of the oracle. This creates a tight-coupling between the design of the filters and the oracle and limits these systems to processing queries over video. In contrast, InQuest decouples the proxy(s) from the oracle, thus enabling users to provide custom models which makes it easy for InQuest to work across different modalities of data.

AQP with DNN-based queries. Recent work has focused on accelerating DNN-based queries over large unstructured datasets in the batch setting [4, 7, 21, 24–27, 35]. While these systems are similar to InQuest in their use of DNNs to answer queries over large unstructured datasets, certain features of their designs make it difficult to adapt them to the streaming setting. For example, NoScope [25] and Tahoma [4] rely on drawing a representative sample from the full dataset before query submission in order to train and validate specialized DNNs and model cascades. ABae [27] and SUPG [26] use sampling techniques over the entire dataset to optimize their oracle sampling strategies. ExSample [35] takes the full dataset and splits it into chunks before query submission in order to perform Thompson sampling [41] across these chunks. These systems would need to be modified substantially to work in the streaming setting, where the dataset is presented to the system in an online fashion.

Proxies in query processing. The use of proxy models for computing cheap approximations spans a variety of use cases in query processing. In certain video analytics systems [11, 32], proxies only compute binary predicates and they are all implemented in a single DNN (potentially at different layers). This is in contrast to our work, in which users provide proxies that can compute arbitrary statistics independent from the oracle. Systems such as NoScope, ABae, and SUPG [25–27] use proxies to estimate query predicates and thereby limit the execution of an expensive oracle to a subset of some large dataset. InQuest uses proxies in a similar manner for processing queries with a predicate. For queries without a predicate InQuest can use a proxy that computes any bounded real-valued estimate, but it will produce better results if the proxy estimate is correlated with the query's statistic of interest.

7 CONCLUSION

In this work we proposed and analyzed InQuest, a system for accelerating aggregation queries over unstructured streams of data with statistical guarantees on query accuracy. We demonstrated significant improvements over streaming and batch setting baselines on a set of real-world video and text datasets. We further showed that InQuest is not sensitive to its parameter settings, that its major components are all crucial for its performance improvements, and that it is resilient to adversarial shifts in the ground-truth stream parameters. We performed a theoretical analysis and showed that InQuest's sample allocation converged to the optimal sample allocation and that its expected error converged to zero at quantitative rates. To the best of our knowledge, this is the first system designed for processing aggregation queries over streams of multiple modalities. Thus, InQuest has the potential to be applied to a wide range of real-world problems, from processing queries over large networks of streaming video cameras to streams of social media posts.

ACKNOWLEDGMENTS

This research was supported in part by affiliate members and other supporters of the Stanford DAWN project—Ant Financial, Facebook, Google, and VMware—as well as Toyota Research Institute, Cisco, SAP, and the NSF under CAREER grant CNS-1651570. This work is also supported in part by the Open Philanthropy project. 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 National Science Foundation. Toyota Research Institute ("TRI") provided funds to assist the authors with their research but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity.

REFERENCES

[1] Charu C Aggarwal. 2006. On biased reservoir sampling in the presence of stream evolution. In *Proceedings of the 32nd international conference on Very large data bases*. 607–618.

[2] Mohammed Al-Kateb, Byung Suk Lee, and X Sean Wang. 2007. Adaptive-size reservoir sampling over data streams. In *19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)*. IEEE, 22–22.

[3] ALERTWildfire. 2022. AlertWildfire. Retrieved Dec. 28, 2022 from https://www.alertwildfire.org/

[4] Michael R. Anderson, Michael Cafarella, German Ros, and Thomas F. Wenisch. 2019. Physical Representation-Based Predicate Optimization for a Visual Analytics Database. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1466–1477. https://doi.org/10.1109/icde.2019.00132

[5] Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Itaru Nishizawa, Justin Rosenstein, and Jennifer Widom. 2003. STREAM: the stanford stream data manager (demonstration description). In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 665–665.

[6] Shivnath Babu and Jennifer Widom. 2001. Continuous queries over data streams. *ACM Sigmod Record* 30, 3 (2001), 109–120.

[7] Favyen Bastani, Songtao He, Arjun Balasingam, Karthik Gopalakrishnan, Mohammad Alizadeh, Hari Balakrishnan, Michael Cafarella, Tim Kraska, and Sam Madden. 2020. MIRIS: Fast Object Track Queries in Video. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1907–1921. https://doi.org/10.1145/3318464.3389692

[8] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. 2017. Enriching Word Vectors with Subword Information. In *Transactions of the Association for Computational Linguistics*, Vol. 5. 135–146. arXiv:1607.04606 http://arxiv.org/abs/1607.04606

[9] Vladimir Braverman and Rafail Ostrovsky. 2013. Generalizing the layering method of indyk and woodruff: Recursive sketches for frequency-based vectors on streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 58–70.

[10] Richard H Browne. 1995. On the use of a pilot sample for sample size determination. *Statistics in medicine* 14, 17 (1995), 1933–1940.

[11] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David G. Andersen, Michael Kaminsky, and Subramanya R. Dulloor. 2019. Scaling Video Analytics on Constrained Edge Nodes. In *Proceedings of the 2nd SysML Conference*. Palo Alto, CA, USA, 12 pages. arXiv:1905.13536 http://arxiv.org/abs/1905.13536

[12] Sirish Chandrasekaran and Michael J Franklin. 2002. Streaming queries over streaming data. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 203–214.

[13] Yueting Chen, Xiaohui Yu, Nick Koudas, and Ziqiang Yu. 2021. Evaluating Temporal Queries Over Video Feeds. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 287–299. https://doi.org/10.1145/3448016.3452803

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[15] AWS EC2. 2023. G4 On-Demand Pricing. Retrieved Apr. 11, 2023 from https://aws.amazon.com/ec2/instance-types/g4/

[16] Apache Flink. 2023. Tumbling Windows. Retrieved May 13, 2023 from https://nightlies.apache.org/flink/flink-docs-master/docs/dev/datastream/operators/windows/#tumbling-windows

[17] Daniel Y. Fu, Will Crichton, James Hong, Xinwei Yao, Haotian Zhang, Anh Truong, Avanika Narayan, Maneesh Agrawala, Christopher Ré, and Kayvon Fatahalian. 2019. Rekall: Specifying Video Events using Compositions of Spatiotemporal Labels. In *SOSP 2019 Workshop on AI Systems*. 16 pages. arXiv:1910.02993 http://arxiv.org/abs/1910.02993

[18] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask R-CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*. 2980–2988. https://doi.org/10.1109/ICCV.2017.322

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[20] James Hong, Will Crichton, Haotian Zhang, Daniel Y. Fu, Jacob Ritchie, Jeremy Barenholtz, Ben Hannel, Xinwei Yao, Michaela Murray, Geraldine Moriba, Maneesh Agrawala, and Kayvon Fatahalian. 2021. Analysis of Faces in a Decade of US Cable TV News. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (Virtual Event, Singapore) *(KDD '21)*. Association for Computing Machinery, New York, NY, USA, 3011–3021. https://doi.org/10.1145/3447548.3467134

[21] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B. Gibbons, and Onur Mutlu. 2018. Focus: Querying Large Video Datasets with Low Latency and Low Cost. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 269–286. https://www.usenix.org/conference/osdi18/presentation/hsieh

[22] HuggingFace. 2022. Twitter-roBERTa-base for Sentiment Analysis. Retrieved Dec. 29, 2022 from https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment-latest

[23] Kaggle. 2022. Customer Support on Twitter. Retrieved Dec. 29, 2022 from https://www.kaggle.com/datasets/thoughtvector/customer-support-on-twitter

[24] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. BlazeIt: Optimizing Declarative Aggregation and Limit Queries for Neural Network-Based Video Analytics. *Proc. VLDB Endow.* 13, 4 (dec 2019), 533–546. https://doi.org/10.14778/3372716.3372725

[25] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Neural Network Queries over Video at Scale. *Proc. VLDB Endow.* 10, 11 (aug 2017), 1586–1597. https://doi.org/10.14778/3137628.3137664

[26] Daniel Kang, Edward Gan, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2020. Approximate Selection with Guarantees Using Proxies. *Proc. VLDB Endow.* 13, 12 (jul 2020), 1990–2003. https://doi.org/10.14778/3407790.3407804

[27] Daniel Kang, John Guibas, Peter Bailis, Tatsunori Hashimoto, Yi Sun, and Matei Zaharia. 2021. Accelerating Approximate Aggregation Queries with Expensive Predicates. *Proc. VLDB Endow.* 14, 11 (jul 2021), 2341–2354. https://doi.org/10.14778/3476249.3476285

[28] Daniel Kang, John Guibas, Peter Bailis, Tatsunori Hashimoto, Yi Sun, and Matei Zaharia. 2021. Proof: Accelerating Approximate Aggregation Queries with Expensive Predicates. Retrieved Dec. 30, 2022 from https://ddkang.github.io/papers/2021/abae-tech-report.pdf

[29] Daniel Kang, John Guibas, Peter D. Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2022. TASTI: Semantic Indexes for Machine Learning-Based Queries over Unstructured Data. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1934–1947. https://doi.org/10.1145/3514221.3517897

[30] Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, and Matei Zaharia. 2020. Jointly Optimizing Preprocessing and Inference for DNN-Based Visual Analytics. *Proc. VLDB Endow.* 14, 2 (oct 2020), 87–100. https://doi.org/10.14778/3425879.3425881

[31] Asterios Katsifodimos and Sebastian Schelter. 2016. Apache Flink: Stream Analytics at Scale. In *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*. 193–193. https://doi.org/10.1109/IC2EW.2016.56

[32] Nick Koudas, Raymond Li, and Ioannis Xarchakos. 2022. Video Monitoring Queries. *IEEE Trans. on Knowl. and Data Eng.* 34, 10 (oct 2022), 5023–5036. https://doi.org/10.1109/TKDE.2020.3048606

[33] Daniel Loureiro, Francesco Barbieri, Leonardo Neves, Luis Espinosa Anke, and Jose Camacho-collados. 2022. TimeLMs: Diachronic Language Models from Twitter. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Association for Computational Linguistics, Dublin, Ireland, 251–260. https://doi.org/10.18653/v1/2022.acl-demo.25

[34] Samuel Madden and Michael J Franklin. 2002. Fjording the stream: An architecture for queries over streaming sensor data. In *Proceedings 18th International Conference on Data Engineering*. IEEE, 555–566.

[35] Oscar Moll, Favyen Bastani, Sam Madden, Mike Stonebraker, Vijay Gadepally, and Tim Kraska. 2022. ExSample: Efficient Searches on Video Repositories through Adaptive Sampling. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 2956–2968. https://doi.org/10.1109/ICDE53745.2022.00266

[36] Penn State Eberly College of Science: Department of Statistics. 2023. Lesson 6: Stratified Sampling. Retrieved Jul. 17, 2023 from https://online.stat.psu.edu/stat506/book/export/html/655

[37] Art Owen and Yi Zhou. 2000. Safe and effective importance sampling. *J. Amer. Statist. Assoc.* 95, 449 (2000), 135–143.

[38] Van L Parsons. 2014. Stratified sampling. *Wiley StatsRef: Statistics Reference Online* (2014), 1–11.

[39] Gregory Piatetsky-Shapiro and Charles Connell. 1984. Accurate Estimation of the Number of Tuples Satisfying a Condition. *SIGMOD Rec.* 14, 2 (jun 1984), 256–276. https://doi.org/10.1145/971697.602294

[40] Alex Poms, Will Crichton, Pat Hanrahan, and Kayvon Fatahalian. 2018. Scanner: Efficient Video Analysis at Scale. *ACM Trans. Graph.* 37, 4, Article 138 (jul 2018), 13 pages. https://doi.org/10.1145/3197517.3201394

[41] Daniel J. Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. 2018. A Tutorial on Thompson Sampling. *Found. Trends Mach. Learn.* 11, 1 (jul 2018), 1–96. https://doi.org/10.1561/2200000070

[42] Matthew Russo, Tatsunori Hashimoto, Daniel Kang, Yi Sun, and Matei Zaharia. 2022. InQuest Technical Report. Retrieved Dec. 30, 2022 from https://github.com/stanford-futuredata/InQuest/blob/main/inquest_technical_report.pdf

[43] Internet Live Stats. 2013. Twitter Usage Statistics. Retrieved Nov. 27, 2022 from https://www.internetlivestats.com/twitter-statistics/

[44] Twitch Tracker. 2022. Twitch broadcast time for all channels by month. Retrieved Nov. 27, 2022 from https://twitchtracker.com/statistics/stream-time

[45] Ioannis Xarchakos and Nick Koudas. 2021. Querying for Interactions. *IEEE Transactions on Knowledge and Data Engineering* (2021), 1–1. https://doi.org/10.1109/TKDE.2021.3094997

[46] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. 2013. Discretized Streams: Fault-Tolerant Streaming Computation at Scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (Farminton, Pennsylvania) *(SOSP '13)*. Association for Computing Machinery, New York, NY, USA, 423–438. https://doi.org/10.1145/2517349.2522737

[47] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 377–392. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zhang

[48] Yuhao Zhang and Arun Kumar. 2020. Panorama: A Data System for Unbounded Vocabulary Querying over Video. *Proc. VLDB Endow.* 13, 4 (jan 2020), 477–491. https://doi.org/10.14778/3372716.3372721