MacroBase: Prioritizing Attention in Fast Data

FIRAS ABUZAID, PETER BAILIS, JIALIN DING, and EDWARD GAN, Stanford InfoLab
SAMUEL MADDEN, MIT CSAIL
DEEPAK NARAYANAN, KEXIN RONG, and SAHAANA SURI, Stanford InfoLab

As data volumes continue to rise, manual inspection is becoming increasingly untenable. In response, we present MacroBase, a data analytics engine that prioritizes end-user attention in high-volume *fast data* streams. MacroBase enables efficient, accurate, and modular analyses that highlight and aggregate important and unusual behavior, acting as a search engine for fast data. MacroBase is able to deliver order-of-magnitude speedups over alternatives by optimizing the combination of explanation (i.e., feature selection) and classification tasks and by leveraging a new reservoir sampler and heavy-hitters sketch specialized for fast data streams. As a result, MacroBase delivers accurate results at speeds of up to 2M events per second per query on a single core. The system has delivered meaningful results in production, including at a telematics company monitoring hundreds of thousands of vehicles.

CCS Concepts: • **Information systems** → **Online analytical processing engines**; *Stream management*;

Additional Key Words and Phrases: Streaming, analytics, database

ACM Reference format:
Firas Abuzaid, Peter Bailis, Jialin Ding, Edward Gan, Samuel Madden, Deepak Narayanan, Kexin Rong, and Sahaana Suri. 2018. MacroBase: Prioritizing Attention in Fast Data. *ACM Trans. Database Syst.* 43, 4, Article 15 (December 2018), 45 pages.
https://doi.org/10.1145/3276463

1 INTRODUCTION

Data volumes are quickly outpacing human abilities to process them. Today, Twitter, LinkedIn, and Facebook each record over 80M events per second [15, 86, 111]. These volumes are growing and are becoming more common: machine-generated data sources such as sensors, processes, and automated systems are projected to increase data volumes by 40% each year [69]. However, human attention remains limited, and manual inspection of these data volumes is increasingly difficult. Today's best-of-class application operators anecdotally report accessing under 6% of data they collect [17], primarily in reactive root-cause analyses.

While humans cannot manually inspect these *fast data* streams, machines can [17]. Machines can filter, highlight, and aggregate fast data, winnowing and summarizing data before it reaches a

Authors' addresses: F. Abuzaid, P. Bailis, J. Ding, and E. Gan, Stanford InfoLab, 353 Serra Mall, Stanford, CA, 94305; emails: fabuzaid@stanford.edu, pbailis@cs.stanford.edu, jding09@stanford.edu, egan1@stanford.edu; S. Madden, MIT CSAIL, 32 Vassar St, Cambridge, MA, 02139; email: madden@csail.mit.edu; D. Narayanan, K. Rong, and S. Suri, Stanford InfoLab, 353 Serra Mall, Stanford, CA, 94305; emails: {deepakn, krong, sahaana}@stanford.edu. Authors listed alphabetically.

ACM 0362-5915/2018/12-ART15 $15.00
https://doi.org/10.1145/3276463

user. As each result shown to the end user consumes their attention [102], we can help prioritize this attention by leveraging computational resources to maximize the utility of each result shown. That is, fast data necessitates a search engine to help identify the most relevant data and trends (and to allow non-expert users to issue queries). The increased availability of elastic computation as well as advances in machine learning and statistics suggest that the construction of such an engine is possible.

However, the design and implementation of this infrastructure remains challenging. Today, application developers and analysts can employ a range of scalable dataflow processing engines to compute over fast data (over 20 in the Apache Software Foundation alone). However, these engines leave the actual implementation of scalable analysis operators that prioritize attention (e.g., highlighting, grouping, and contextualizing important behaviors) up to the application developer. This development is hard: fast data analyses must (i) limit their results to avoid overwhelming user attention while (ii) executing quickly to keep up with immense data volumes and (iii) adapting to changes within the data stream itself. Thus, designing and implementing these analytics operators requires a combination of domain expertise, statistics and machine learning, and dataflow processing. This combination is rare. Instead, today's high-end industrial deployments overwhelmingly rely on a combination of static rules and thresholds that analysts report are computationally efficient but brittle and error-prone; manual analysis is typically limited to reactive, post-hoc error diagnosis that can take hours to days.

To bridge this gap between the availability of low-level dataflow processing engines and the need for efficient, accurate analytics, we have begun the development of MacroBase, a fast data analysis system. The core concept behind MacroBase is simple: to prioritize attention, an analytics engine should provide operators that automatically classify and explain fast data volumes to users. MacroBase executes extensible streaming dataflow pipelines that contain operators for both *classifying* individual data points and *explaining* groups of points by aggregating them and highlighting commonalities of interest (i.e., via feature selection). Combined, these operators ensure that the returned results capture the most important properties of data. As in conventional relational analytics, the design of a core set of efficient operators generalizes across application domains.

The resulting research challenge is to determine this efficient, accurate, and modular set of core classification and explanation operators for prioritizing attention in fast data. The statistics and machine learning literature is replete with candidate algorithms, but it is unclear which can execute online at fast data volumes, and, more importantly, how these operators can be composed in an end-to-end system. Thus, in this article, we both introduce the core MacroBase architecture—which combines domain-specific feature extraction with streaming classification and explanation operators—and present the design and implementation of MacroBase's default streaming classification and explanation operators. In the absence of labeled training data, MacroBase executes operators for unsupervised, density-based classification that highlight points lying far from the overall population according to user-specified *metrics* of interest (e.g., power drain). MacroBase subsequently executes sketch-based explanation operators, which highlight correlations that most differentiate outlying data points according to their *attributes* (e.g., firmware version, device ID).

Classification and specifically anomaly detection is a well-studied problem in machine learning and data mining [8, 44]: much work has been done to study how best to detect anomalies in a variety of domains and data modalities, such as network traffic [12], graph data [13], time series data [77], and video [34]. While MacroBase incorporates its own default classification operators for labeling points as "outliers" and "inliers," due to our modular design, we can incorporate more sophisticated classification techniques based on user's needs. MacroBase combines this classification with an explanation step, thus providing an *explanation* engine that automatically generates explanations for statistically unusual data points.

Users of the open source MacroBase prototype[1] have utilized MacroBase's classification and explanation operators to find unusual, previously unknown behaviors in fast data from mobile devices, datacenter telemetry, vehicles, and manufacturing processes, as in the following example.

Example. A mobile application manufacturer issues a MacroBase query to monitor power drain readings (i.e., *metrics*) across devices and application versions (i.e., *attributes*). MacroBase's default operator pipeline reports that devices of type B264 running application version 2.26.3 are 60 times more likely to experience abnormally high power drain than the rest of the stream, indicating a potential problem with the interaction between devices of type B264 and application version 2.26.3.

Beyond this basic default functionality, MacroBase allows users to tune their queries by (i) adding domain-specific feature transformations (e.g., time-series operations such as Fourier transform and autocorrelation) to their pipelines—without modifying later pipeline steps, (ii) providing supervised classification rules (or labels) to complement or replace unsupervised classifiers, and (iii) authoring custom streaming transformation, classification, and explanation operators, whose interoperability is enforced by MacroBase's type system and can be combined with relational operators.

Example. The mobile application developer also wishes to find time-varying power spikes within the stream, so she reconfigures her pipeline by adding a time-series feature transformation to identify time periods with abnormal time-varying frequencies. She later adds a manual rule to capture all readings with power drain greater than 100W and a custom time-series explanation operator [76]—all without modifying the remainder of the operator pipeline.

Developing these operators necessitated several algorithmic advances, which we address as core research challenges in this article.

To provide responsive analyses over dynamic data sources, MacroBase's default operators are designed to adapt to shifts in data. MacroBase leverages a novel stream sampler, called the *Adaptable Damped Reservoir* (ADR), which performs sampling over arbitrarily sized, exponentially damped windows. MacroBase uses the ADR to incrementally train unsupervised classifiers based on statistical density estimation that can identify typical behavioral modes even in the presence of extreme data points [65]. MacroBase also adopts exponentially weighted sketching and streaming data structures [38, 106] to track correlations between attribute-value pairs, improving responsiveness and accuracy in explanation.

To provide interpretable explanations of correlations across classified points, MacroBase adopts a metric from epidemiology called the *relative risk ratio* that describes the relative occurrence of key attributes (e.g., age, sex) among the "outlier" and "inlier" populations. In computing this statistic, MacroBase employs two new optimizations. First, MacroBase exploits the cardinality imbalance between classified points to accelerate explanation generation, an optimization enabled by the combination of detection and explanation. Instead of inspecting outliers and inliers separately, MacroBase first examines the small set of outliers, then aggressively prunes its search space over the much larger set of inliers. Second, MacroBase exploits the fact that many fast data streams contain repeated measurements from devices with similar attributes (e.g., firmware version) during risk ratio computation, reducing data structure maintenance overhead via a new counting sketch, the *Amortized Maintenance Counter* (AMC). These optimizations improve performance while highlighting the subset of attributes that matter most.

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

We report on early production experiences and quantitatively evaluate MacroBase's performance and accuracy on both production telematics data as well as a range of publicly available real-world datasets. MacroBase exhibits order-of-magnitude performance improvements over existing operators at rates of up to 2M events per second per query while delivering accurate results in controlled studies using both synthetic and real-world data. As we discuss, this ability to quickly process large data volumes can also improve result quality: larger datasets are more resilient to the multiple hypothesis testing problem [96], and improve result significance. We also demonstrate MacroBase's extensibility via case studies in mobile telematics, electricity metering, and video-based surveillance, and integrate MacroBase with several existing analytics frameworks.

We make the following contributions in this article:

- MacroBase, an analytics engine and architecture for analyzing fast data streams that is the first to combine streaming outlier detection and streaming data explanation.
- The Adaptable Damped Reservoir, the first exponentially damped reservoir sample to operate over arbitrary windows, which MacroBase leverages in online classifier training.
- An optimization for improving the efficiency of combined detection and explanation by exploiting cardinality imbalance between classes in streams.
- The Amortized Maintenance Counter, a new heavy-hitters sketch that allows fast updates by amortizing sketch pruning across multiple observations of the same item.
- Adaptations to the core MacroBase pipeline that support operation over aggregated data, sliding-window explanation, support for alternative explanation metrics, and lightweight sample-based feature exploration.

The remainder of this article proceeds as follows. Section 2 describes our target environment by presenting motivating use cases. Section 3 presents the MacroBase's interaction model and primary default analysis pipeline (which we denote MDP). Section 4 describes MacroBase's default streaming classification operators and presents the ADR sampler. Section 5 describes MacroBase's default streaming explanation operator, including its cardinality-aware optimization and the AMC sketch. We experimentally evaluate MacroBase's accuracy and runtime, report on experiences in production, and demonstrate extensibility via case studies in Section 7. Section 8 describes extensions and modifications to MacroBase's default analysis pipeline inspired by user demands, and Section 9 reports lessons from MacroBase prototype usage. Section 10 discusses related work, and Section 11 concludes.

This invited article includes material from [16] and, as discussed in Section 10, significantly extends this earlier article.

2 TARGET ENVIRONMENT

MacroBase provides application writers and system analysts an end-to-end analytics engine capable of classifying data within high-volume streams while highlighting important properties of the data within each class. As examples of the types of workloads we seek to support, we draw on three motivating use cases from industry.

Mobile Applications. Cambridge Mobile Telematics (CMT) is a 5-year-old telematics company whose mission is to make roads safer by making drivers more aware of their driving habits. CMT provides drivers with a smartphone application and mobile sensor for their vehicles, and collects telemetry data from many hundreds of thousands of vehicles at rates of tens of Hz. CMT uses this data to provide users with feedback about their driving.

CMT's engineers report that monitoring their application has proven especially challenging. CMT's operators, who include database and systems research veterans, report difficulty in answering several questions: Is the CMT application behaving as expected? Are all users able to upload and view their trips? Are sensors operating at a sufficiently granular rate and in a power-efficient manner? The most severe problems in the CMT application are caught by quality assurance and customer service, but many behaviors are more pernicious. For example, Apple iOS 9.0 beta 1 introduced a buggy Bluetooth stack that prevented iOS devices from connecting to CMT's sensors. Few devices ran these versions, so the overall failure rate was low; as a result, CMT's data volume and heterogeneous install base (which includes the 24K distinct device types in the Android ecosystem) obscured a potentially serious widespread issue in later releases of the application. Given low storage costs, CMT records all of the data required to perform analytic monitoring to detect such behaviors, yet CMT's engineers report they have lacked a solution for doing so in a timely and efficient manner.

In this article, we report on our experiences deploying MacroBase at CMT, where the system has highlighted interesting production behaviors such as those above.

Datacenter Operation. Datacenter and server operation represents one of the highest-volume data sources today. Engineers at Twitter and Linkedin report challenges while monitoring their billion-plus events per minute data volumes in order to identify misbehaving servers, applications, and virtual machines.

For example, Amazon AWS recently suffered a failure in its DynamoDB service, resulting in outages at sites including Netflix and Reddit. Amazon engineers reported that "after we addressed the key issue . . . we were left with a low overall error rate, hovering between 0.15–0.25%. We knew there would be some cleanup to do after the event," and therefore the engineers deferred maintenance. However, the engineers "did not realize soon enough that this low overall error rate was giving some customers disproportionately high error rates" due to a misbehaving server partition [2].

This public postmortem is representative of many scenarios described by system operators in interviews. At a major social network, engineers reported that the challenge of identifying transient slowdowns and failures across hosts and containers is exacerbated by the heterogeneity of workload tasks. Thus, failure postmortems can involve hours to days of manual analysis across each workload type. Unlike the CMT use case, we do not directly present results over production data from these scenarios. However, monitoring datacenter telemetry is an active area of research within the MacroBase project.

Industrial Monitoring. Increased sensor availability has spurred interest in and collection of fast data in industrial deployments. These deployments often already rely on legacy monitoring systems, but we encountered several operators who reported a desire for analytics and alerting that can adapt to new sensors and changing conditions. These conditions have important consequences. For example, an explosion and fire in July 2010 killed two workers at Horsehead Holding Corp.'s Monaca, PA, zinc manufacturing plant. The US Chemical Safety board's postmortem revealed that "the high rate-of-change alarm warned that the [plant] was in imminent danger 10 minutes before it exploded, but there appears to have been no specific alarm to draw attention of the operator to the subtle but dangerous temperature changes that were taking place much (i.e., hours) earlier." The auditor noted that "it should be possible to design a more modern control system that could draw attention to trends that are potentially hazardous" [67].

In this article, we illustrate the potential to highlight unusual behaviors within electrical utilities.

3 MACROBASE ARCHITECTURE AND APIS

As a fast data analytics engine, MacroBase filters and aggregates large, high-volume streams of potentially heterogeneous data. As a result, MacroBase's architecture is designed for high-performance execution as well as flexible operation across many domains, using an array of classification and explanation operators. In this section, we describe MacroBase's query processing architecture, approach to extensibility, and interaction modes.

3.1 Core Concepts

To prioritize attention, MacroBase executes streaming analytics operators that help filter and aggregate the stream by combining two classes of operators: Classification and Explanation.

Classification. Classification operators score or label individual data points according to user-specified classes. For example, MacroBase can classify an input stream of power drain readings into points representing statistically "normal" readings and abnormal "outlying" readings. A MacroBase classification operator must assign a binary label to each data point. We refer to these labels as the "inlier" and "outlier" classes corresponding to normal behavior and abnormal behavior, and MacroBase will seek to summarize differences between the two classes.

At scale, surfacing even a handful of raw data points per second can overwhelm end users, especially if each data point contains multidimensional and/or categorical information. As a result, MacroBase employs a second type of operator:

Explanation. Explanation operators aggregate and group multiple data points based on the "outlier" and "inlier" labels produced in the previous Classification step. For example, MacroBase can describe commonalities among points in the outliers, as well as differences between the outliers and inliers. Each result returned by an explanation operator can represent many individual classification outputs, further prioritizing attention.

As we discuss in Section 10, classification and explanation are core topics in several communities, including statistics and machine learning. Our goal in developing MacroBase is to design core operators for each task that execute quickly over streaming data that may change over time and can be composed as part of larger end-to-end pipelines. Conventional relational analytics provide a well-defined set of composable, reusable operators; despite pressing application demands at scale, the same cannot be said of classification and explanation today. Identifying these operators and combining them with appropriate domain-specific feature extraction operators enables reuse beyond one-off, ad-hoc analyses.

Thematically, our focus is on developing operators that deliver more information using less output. This score-and-aggregate strategy is reminiscent of many data-intensive domains, including search. When employed in a system designed for extensibility, a small number of optimized, composable operators can execute across domains. However, as we show, adapting these operators for efficient, extensible fast data pipelines requires design modification and even enables new optimizations.

3.2 System Architecture

Query Pipelines. MacroBase executes pipelines of specialized dataflow operators over input data streams. Each MacroBase *query* specifies a set of input data sources as well as a logical query plan, or *pipeline* of streaming operators, that describes the analysis.

MacroBase's pipeline architecture is guided by two principles. First, all operators operate over streams. Batch execution is supported by streaming over stored data. Second, MacroBase uses the compiler's type system to enforce interoperability. Each operator must implement one of several type signatures (shown in Table 1). In turn, the compiler enforces that all pipelines composed of these operators will adhere to the common structure we describe below.

Table 1. MacroBase's Core Data and Operator Types

DATA TYPES	
Point := (`array<double>` metrics, `array<varchar>` attributes)	
Explanation := (`array<varchar>` attributes, `array<double>` statistics)	
OPERATOR INTERFACE	
Operator	*Type Signature*
Ingestor	external data source(s) → `stream<`*Point*`>`
Transformer	`stream<`*Point*`>` → `stream<`*Point*`>`
Classifier	`stream<`*Point*`>` → `stream<(label, `*Point*`)>`
Explainer	`stream<(label, `*Point*`)>` → `stream<`*Explanation*`>`
Pipeline	Ingestor → `stream<`*Explanation*`>`

Each operator implements a typed, stream-oriented dataflow interface. A pipeline can contain multiple transformation operators and ultimately returns a single stream of explanations.

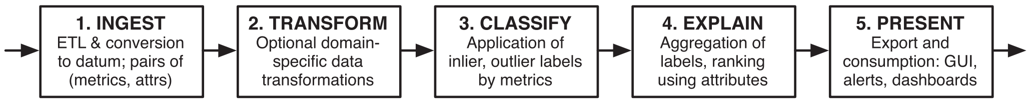

1. INGEST
ETL & conversion to datum; pairs of (metrics, attrs)
2. TRANSFORM
Optional domain-specific data transformations
3. CLASSIFY
Application of inlier, outlier labels by metrics
4. EXPLAIN
Aggregation of labels, ranking using attributes
5. PRESENT
Export and consumption: GUI, alerts, dashboards

Fig. 1. MacroBase's default analytics pipeline: MacroBase ingests streaming data as a series of points, which are scored and classified, aggregated by an explanation operator, then ranked and presented to end users.

This *architecture via typing* strikes a balance between the elegance of more declarative but often less flexible interfaces and the expressiveness of more imperative but often less composable interfaces. More specifically, this use of the type system facilitates three important kinds of interoperability. First, users can substitute streaming detection and explanation operators without concern for their interoperability. Early versions of the MacroBase prototype that lacked this modularity were hard to adapt. Second, users can write a range of domain-specific feature transformation operators to perform advanced processing (e.g., time-series operations) without requiring expertise in classification or explanation. Third, MacroBase's operators preserve compatibility with dataflow operators found in traditional stream processing engines. For example, a MacroBase pipeline can contain standard selection, project, join, windowing, aggregation, and group-by operators.

A MacroBase pipeline, illustrated in Figure 1, is structured as follows:

(1) Ingestion. MacroBase ingests data streams for analysis from a number of external data sources. For example, MacroBase's JDBC interface allows users to specify columns of interest from a base view defined by a SQL query. MacroBase subsequently reads the result-set from the JDBC connector, and constructs the set of data points to process, with one point per row in the view. MacroBase currently requires that any necessary stream ordering and joins be performed by this initial ingestion operator.

Each data point contains a set of *metrics*, corresponding to key measurements (e.g., trip time, battery drain), and *attributes*, corresponding to associated metadata (e.g., user ID and device ID). MacroBase uses metrics to detect abnormal or unusual events, and attributes to explain behaviors. In this article, we consider real-valued metrics and categorical attributes.[2]

As an example, to detect the OS version problem at CMT, trip times could be used as a metric, and device and OS type as attributes. To detect the outages at DynamoDB, error rates could be used as a metric, and server or IP address as an attribute. To detect the Horsehead pressure losses,

[2]We discretize continuous attributes (e.g., see [113]) and provide two examples of discretization in Section 7.4.

pressure gauge readings could be used as metrics and their locations as attributes, as part of an autocorrelation-enabled time-series pipeline (Section 7.4). Today, selecting attributes, metrics, and a pipeline is a user-initiated process; ongoing extensions (Section 11) seek to automate this.

(2) Feature Transformation. Following ingestion, MacroBase executes an optional series of domain-specific data transformations over the stream, which could include time-series–specific operations (e.g., windowing, seasonality removal, autocorrelation, frequency analysis), statistical operations (e.g., normalization, dimensionality reduction), and datatype-specific operations (e.g., hue extraction for images, optical flow for video). For example, in Section 7.4, we execute a pipeline containing a grouped Fourier transform operator that aggregates the stream into hour-long windows, then outputs a stream containing the 20 lowest Fourier coefficients for each window as metrics and properties of the window time (hour of day, month) as attributes. Placing this feature transformation functionality at the start of the pipeline allows users to encode domain-specific analyses without modifying later stages. The base type of the stream is unchanged (*Point* $\rightarrow$ *Point*), allowing transforms to be chained. For specialized data types like video frames, operators can subclass *Point* to further increase the specificity of types (e.g., *VideoFramePoint*).

(3) Classification. Following transformation, MacroBase performs classification, labeling each *Point* according to its input metrics as either an outlier or an inlier. Both training and evaluating classifiers on the metrics in the incoming data stream occur in this stage. MacroBase supports a range of models, which we describe in Section 7. The simplest include rule-based models, which check specific metrics for particular values (e.g., if the `Point` metric's L_2-norm is greater than a fixed constant). In Section 4, we describe MacroBase's default unsupervised models, which perform density-based classification into outlier and inlier classes. Users can also use operators that make use of supervised and pre-trained models. Independent of model type, each classifier returns a stream of labeled *Point* outputs (*Point* $\rightarrow$ `(label,` *Point*`)`).

(4) Explanation. Rather than returning all labeled data points, MacroBase aggregates the stream of labeled data points by generating *explanations*. As we describe in detail in Section 5, MacroBase's default pipeline returns explanations in the form of attribute-value combinations (e.g., device ID 5052) that are common among outlier points but uncommon among inlier points. For example, at CMT, MacroBase could highlight devices that were found in at least 0.1% of outlier trips and were at least three times more common among outliers than inliers. Each explanation operator returns a stream of these aggregates (`(label,` *Point*`)` $\rightarrow$ *Explanation*), and explanation operators can subclass *Explanation* to provide additional information, such as statistics about the explanation or representative sequences of points to contextualize time-series outliers.

Because MacroBase processes streaming data, explanation operators continuously summarize the stream. However, continuously emitting explanations may be wasteful if users only need explanations at the granularity of seconds, minutes, or longer. As a result, MacroBase's explanation operators are designed to emit explanations on demand, either in response to a user request, or in response to a periodic timer. In this way, explanation operators act as streaming view maintainers.

(5) Presentation. The number of output explanations may still be large. As a result, most pipelines rank explanations by statistics specific to the explanations before presentation. By default, MacroBase delivers a ranked list of explanations—sorted by their degree of outlier—occurrence to downstream consumers. MacroBase's default presentation mode is a static report rendered via a REST API or GUI. In the former, programmatic consumers (e.g., reporting tools such as PagerDuty) can automatically forward explanations to downstream reporting or operational systems. In the GUI, users can interactively inspect explanations and iteratively define their MacroBase queries.

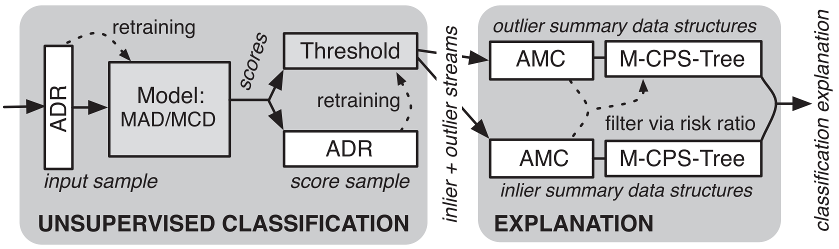


Fig. 2. MDP: MacroBase's default streaming classification (Section 4) and explanation (Section 5) operators.

In practice, we have found that GUI-based exploration is an important first step in formulating standing MacroBase queries that can later be used in production.

Extensibility. As we discussed in Section 1 and demonstrate in Section 7.4, MacroBase's pipeline architecture lends itself to three major means of extensibility. First, users can add new domain-specific feature transformations to the start of a pipeline without modifying the rest of the pipeline. Second, users can input rules and/or labels to MacroBase to perform supervised classification. Third, users can write their own feature transformation, classification, and explanation operators, as well as new pipelines. This third option is the most labor-intensive, but is also the interface with which MacroBase's maintainers author new pipelines. These interfaces have proven useful to non-experts: a master's student at Stanford and a master's student at MIT each implemented and tested a new outlier detector operator in less than a week of part-time work, and MacroBase's core maintainers currently require less than an afternoon of work to author and test a new pipeline.

By providing a set of interfaces with which to extend pipelines (with varying expertise required), MacroBase places emphasis on "pay as you go" deployment [17]. MacroBase's Default Pipeline (MDP, which we illustrate in Figure 2 and describe in the following two sections) is optimized for efficient, accurate execution over a variety of data types without relying on labeled data or rules. It foregoes domain-specific feature extraction and instead operates directly on raw input metrics. However, as we illustrate in Section 7.4, this interface design enables users to incorporate more sophisticated features such as domain-specific feature transformation, time-series analysis, and supervised models.

In this article, we present MacroBase's interfaces using an object-oriented interface, reflecting their current implementation. However, each of MacroBase's operator types is compatible with existing stream-to-relation semantics [14], theoretically allowing additional relational and stream-based processing between stages. Realizing this mapping and the potential for higher-level declarative interfaces above MacroBase's pipelines are promising areas for future work.

Operating Modes. MacroBase supports three operating modes. First, MacroBase's graphical front-end allows users to interactively explore their data by configuring different inputs and selecting different combinations of metrics and attributes. This is typically the first step in interacting with the engine. Second, MacroBase can execute one-shot queries that can programmatically run over a single pass of the data. Third, MacroBase can execute streaming queries that can be run programmatically over a potentially infinite stream of data. In streaming mode, MacroBase continuously ingests data points and supports exponentially decaying averages that give precedence to more recent points (e.g., decreasing the importance of points at a rate of 50% every hour). MacroBase continuously re-renders query results, and if desired, triggers automated alerting for downstream consumers.

4 MDP CLASSIFICATION

MacroBase's classification operators separate input data points into inliers and outliers. While MacroBase allows users to configure their own classification operators, MacroBase's default MDP operators use robust estimation procedures to fit a distribution to the input data and identify the least likely points within the distribution using quantile estimation. To enable streaming execution, we introduce the Adaptable Damped Reservoir, which MacroBase uses for model retraining and quantile estimation.

4.1 Robust Distribution Estimation

MDP relies on unsupervised density-based classification to estimate the data distribution and identify points that are abnormal relative to this distribution (i.e., are in the 99th percentile, or are in low-density regions of the distribution). However, for certain methods following this framework, a small number of anomalous points can have a large impact on parameter estimation, and by extension, the datapoints that MacroBase classifies as outliers. As an example, consider the *Z-Score* of a point drawn from a univariate sample, which measures the number of standard deviations that the point lies away from the sample mean. This provides a normalized way to measure the "outlying"-ness of a point (e.g., a Z-Score of three indicates the point lies three standard deviations from the mean). However, the Z-Score, which inherently assumes a Gaussian data distribution, is not robust to outliers: a single outlying value can skew the mean and standard deviation by an unbounded amount, limiting its utility.

To address this challenge, MacroBase's MDP pipeline leverages robust statistical estimation [65], a branch of statistics that computes statistical distributions for data that is mostly well-behaved but may contain ill-behaved or anomalous data points. Given a distribution that reliably fits most of the data, we can then measure each point's distance from this distribution to find outliers [79].

For univariate data, a robust variant of the Z-Score-based method is to use the median and the Median Absolute Deviation (MAD) in place of mean and standard deviation as measures of the location and scatter of the distribution. The MAD measures the median of the absolute distance from each point in the sample to the sample median. Since the median itself is resistant to outliers, each outlying data point has limited impact on the MAD score of all other points in the sample.

For multivariate data, the Minimum Covariance Determinant (MCD) provides similar robust estimates for location and spread [66]. The MCD estimator finds the tightest group of points that best represents a sample, and summarizes the set of points according to its location μ and scatter C (i.e., covariance) in metric space. Given these estimates, we can compute the distance between a point x and the distribution via the *Mahalanobis distance* $\sqrt{(x-\mu)^T C^{-1}(x-\mu)}$; intuitively, the Mahalanobis distance normalizes (or warps) the metric space via the scatter and then measures the distance to the center of the transformed space using the mean.

As Figure 3 empirically demonstrates, MAD and MCD reliably classify inlying and outlying data points despite increasing outlier contamination. We examine a dataset of 10M points drawn from two distributions: a uniform *inlier* distribution, with radius 50 centered at the origin, and a uniform *outlier* distribution, with radius 50 centered at (1,000, 1,000). We vary the proportion of outlying points in each to evaluate the effect on the Z-Score, MAD, and MCD-based classifiers, using univariate points for Z-Score and MAD. While MAD and MCD are resilient to contamination up to 50% (with unchanged F1 score), the Z-Score is unable to distinguish inliers and outliers under even modest contamination.

MCD Implementation. Computing the exact MCD requires examining all subsets of points to find the subset whose covariance matrix exhibits the minimum determinant. This is computationally intractable for even modestly sized datasets. Instead, MacroBase adopts an iterative

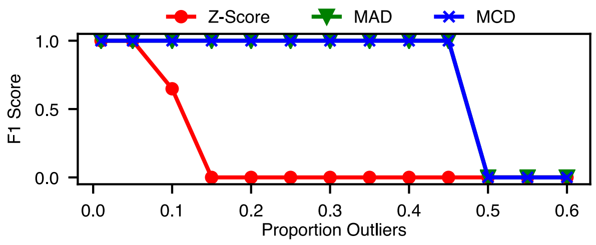


Fig. 3. Accuracy of estimators under contamination by outliers, characterized by the F1 score (high scores better). Robust methods (MCD, MAD) outperform the Z-score-based approach.

approximation called FastMCD [92]. In FastMCD, an initial subset of points S_0 is chosen from the input set of points P. FastMCD computes the covariance C_0 and mean μ_0 of S_0, then performs a "C-step" by finding the set S_1 of points in P that have the $|S_1|$ closest Mahalanobis distances (to C_0 and μ_0). FastMCD subsequently repeats C-steps (i.e., computes the covariance C_1 and mean μ_1 of S_1, selects a new subset S_2 of points in P, and repeats) until the change in the determinant of the sample covariance converges (i.e., $det(S_{i-1}) - det(S_i) < \epsilon$, for small ϵ). To determine which dimensions are most anomalous when using MCD, MacroBase uses the corr-max transformation [52].

Classifying Outliers. Given a query with a single, univariate metric, MDP uses a MAD-based detector, and given a query with multiple metrics, MacroBase computes the MCD using FastMCD [92]. These unsupervised models allow MDP to score points without requiring labels or rules from users. Subsequently, MDP uses a percentile-based cutoff over scores to identify the most extreme points in the sample. Points with scores above the percentile-based cutoff are classified as outliers, reflecting their distance from the body of the distribution.

As a notable caveat, MAD and MCD are parametric estimators, assigning scores while assuming the data is normally distributed, thus identifying point-level extreme deviations from the overall population. A key limitation of these techniques is that they are unable to pinpoint unusual behavior that occurs as a result of higher-level data interactions and correlations, but instead focus only on global outliers. For instance, the default classification operators will be unable to identify contextual outliers such as temporal shifts and behavior, or collective outliers, that become evident only once data is stratified or segmented. As we empirically demonstrate, many interesting behaviors manifest as extreme, global devations, thus robustly locating the center of a population—while ignoring local, small-scale deviations in the distribution's body—suffices to identify many important outlier classes in the applications we study (cf. [60]).

However, the MacroBase architecture can accommodate custom outlier detection techniques from the large body of work on contextual or collective outlier detection, as well as capturing multi-modal behavior [59]. Substituting these application-specific classification operators enables MacroBase to extend to additional domains at the expense of computational efficiency. For instance, we have used Kernel Density Estimation for unsupervised density estimation prior to percentile classification, to identify data points that are not extreme, but are present in low-density regions of the distribution. Additionally, as we demonstrate in Section 7.4, by making use of custom feature transformations (such as Fourier Transformations in conjunction with temporal windowing), MacroBase can provide contextual outlier detection, highlighting regions in time with abnormal power usage. In Section 10, we further discuss alternative outlier classification techniques in more detail, such as one-class SVMs, Hierarchical Temporal Memory, RANSAC, or robust auto encoders [11, 48, 99, 115].

4.2 MDP Streaming Execution

Despite their utility, we are not aware of an existing algorithm for training MAD or MCD in a streaming context.[3] This is especially problematic because, as the distributions within data streams change over time, MDP's estimators should be updated to reflect the change.

ADR. MDP's solution to the retraining problem is a novel adaptation of reservoir sampling over streams, which we call the Adaptable Damped Reservoir (ADR). The ADR maintains a sample of input data that is exponentially weighted toward more recent points; the key difference from traditional reservoir sampling is that the ADR operates over *arbitrary* window sizes, allowing greater flexibility than existing damped samplers. As Figure 2 illustrates, MDP maintains an ADR sample of the input to periodically recompute its robust estimator and a second ADR sample of the outlier scores to periodically recompute its quantile threshold.

The classic reservoir sampling technique can be used to select a uniform sample over a set of data using finite space and one pass [107]. The probability of insertion into the sample, or "reservoir," is inversely proportional to the number of points observed thus far. In the context of stream sampling, we can treat the stream as an infinitely long set of points and the reservoir as a uniform sample over the data observed so far.

In MacroBase, we wish to promptly reflect changes in the data stream. Hence, we adapt a *weighted* sampling approach, in which the probability of data retention decays over time. The literature contains several existing algorithms for weighted reservoir sampling [5, 31, 41]. Most recently, Aggarwal described how to perform exponentially weighted sampling on a per-record basis: that is, the probability of insertion is an exponentially weighted function of the number of points observed so far [5]. While this is useful, as we demonstrate in Section 7, under workloads with variable arrival rates, we may wish to employ a decay policy that decays in *time*, not in number of tuples; specifically, tuple-at-a-time decay may skew the reservoir toward periods of high stream volume.

To support more flexible reservoir behavior, MacroBase adapts an earlier variant of weighted reservoir sampling due to Chao [31, 41] to provide the first exponentially decayed reservoir sampler that decays over arbitrary decay intervals. We call this variant the *Adaptable Damped Reservoir*, or ADR (Algorithm 1). In contrast with existing approaches that decay on a per-tuple basis, the ADR separates the insertion process from the decay decision, allowing both time-based and tuple-based decay policies. Specifically, the ADR maintains a running count c_w of items inserted into the reservoir (of size k) so far. When an item is inserted, c_w is incremented by one (or an arbitrary weight, if desired). With probability $\frac{k}{c_w}$, the item is placed into the reservoir and a random item is evicted from the reservoir. When the ADR is decayed (e.g., via a periodic timer or tuple count), its running count is multiplied by a decay factor (i.e., $c_w := (1 - \alpha)c_w$).

MacroBase currently supports two decay policies.

Time-Based Decay. MacroBase decays the reservoir at a pre-specified rate measured according to real time by computing a uniform sample per decay period, with decay across periods. This is achieved by maintaining an ADR for the stream contents from all prior periods and a regular, uniform reservoir sample for the current period. At the end of the period, the period sample can be inserted into the ADR.

Batch-Based Decay. MacroBase decays the reservoir at a pre-specified rate measured by arbitrarily sized batches of data points, reflecting the effect of variable tuple arrival rates. This is achieved

[3]Specifically, MAD requires computing the median of median distances, meaning streaming quantile estimation alone is insufficient. FastMCD is an inherently iterative algorithm that iteratively re-sorts data.

ALGORITHM 1: ADR: Adaptable Damped Reservoir

```
given k: reservoir size $\in \mathbb{N}$; r: decay rate $\in (0, 1)$;
initialization reservoir $R \leftarrow \{\}$; current weight $c_w \leftarrow 0$;
observe(x: point, w: weight)
    $c_w \leftarrow c_w + w$;
    if $|R| < k$ then
        $R \leftarrow R \cup \{x\}$;
    else
        with probability $\frac{k}{c_w}$, remove random element from $R$ and add $x$ to $R$;
    end
decay()
    $c_w \leftarrow r \cdot c_w$;
```

by computing a uniform sample over time, with decay according to time. In this setting, given a sampling period (e.g., 1s), for each period, insert the average of all points into the ADR.

The validity of this procedure follows from Chao's sampler, which otherwise requires the user to manually manage weights and decay. As in Chao's sampler, in the event of extreme decay, "overweight" items with relative insertion probability $\frac{k}{c_w} > 1$ are always retained in the reservoir until their insertion probability falls below 1, at which point they are inserted normally.

MacroBase's MDP uses the ADR to solve the model retraining and quantile estimations problems.

Maintaining Training Inputs. Either on a tuple-based or time-based interval, MDP retrains models using the contents of an ADR that samples the input data stream. This streaming robust estimator maintenance and evaluation strategy is the first of which we are aware. We discuss this procedure's statistical impact in Section 7.5.

Maintaining Percentile Thresholds. While streaming quantile estimation is well studied, we were unable to find many computationally inexpensive options for an exponentially damped model with arbitrary window sizes. Thus, MacroBase uses an ADR to sample the outlier scores produced by MAD and MCD. The ADR maintains an exponentially damped sample of the scores, which it uses to periodically compute the appropriate score quantile value (e.g., 99th percentile of scores).[4] A sample of size $O(\frac{1}{\epsilon^2}\log(\frac{1}{\delta}))$ yields an ϵ-approximation of an arbitrary quantile with probability $1 - \delta$ [23], so an ADR of size 20K provides an $\epsilon = 1\%$ approximation with 99% probability ($\delta = 1\%$).

In the evaluation (Section 7.3), we show that the proposed ADR enables more robust outlier detection under shifts in the input distribution, or concept drift, compared to alternative reservoir sampling techniques: ADR is able to quickly adapt to systematic shifts in the inlier population, as well as temporary changes when outliers return to the inlier distribution. We defer further discussions regarding concept drifts in date streams in the related work section (Section 10).

5 MDP EXPLANATION

MDP's explanation operators produce explanations to contextualize and differentiate inliers and outliers according to their attributes. In this section, we discuss how MacroBase performs this task by leveraging a metric from epidemiology, the *relative risk ratio* (risk ratio), using a range of data structures. We again begin with a discussion of MDP's batch-oriented operation and introduce a

[4]This enables a simple mechanism for detecting quantile drift: if the proportion of outlier points significantly deviates from the target percentile (i.e., via application of a binomial proportion confidence interval), MDP should recompute the quantile.

ALGORITHM 2: MDP's Outlier-Aware Explanation Strategy

```
  given minimum risk ratio $r$, minimum support $s$, set of outliers $O$, set of inliers $I$;
1 find single attributes w/ support $\geq s$ in $O$ with risk ratio $\geq r$;
2 mine FP-tree over $O$ using only attributes from (1);
3 filter (2) by removing attribute combinations w/ risk ratio $< r$; return;
```

cardinality-based optimization, then discuss how MacroBase executes streaming explanation via the Amortized Maintenance Counter sketch. We end with a discussion of confidence intervals on MDP explanations as well as quality improvements achievable by processing large data volumes.

5.1 Semantics: Support and Risk Ratio

MacroBase produces explanations that describe attributes common to outliers but relatively uncommon to inliers. To identify combinations of attribute values that are relatively common in outliers, MDP finds combinations with high *risk ratio (or relative risk ratio).* This ratio is a standard diagnostic measure used in epidemiology, and is used to determine potential causes for disease [82]. Formally, given an attribute combination appearing a_o times in the outliers and a_i times in the inliers, where there are b_o other outliers and b_i other inliers, the risk ratio is defined as

$$\text{risk ratio} = \frac{a_o/(a_o + a_i)}{b_o/(b_o + b_i)}.$$

Intuitively, the risk ratio quantifies how much more likely a data point is to be an outlier if it matches a specific attribute combination, as opposed to the general population. To eliminate explanations corresponding to rare but non-systemic combinations, MDP finds combinations with high *support*, or occurrence (by relative count) in outliers. To facilitate these two tests, MDP accepts a minimum risk ratio and level of outlier support as input parameters. As an example, *MDP* may find that 500 of 890 records flagged as outliers correspond to iPhone X devices (outlier support of 56.2%), but, if 80,191 of 90,922 records flagged as inliers also correspond to iPhone X devices (inlier support of 88.2%), we are likely uninterested in iPhone X as it has a low risk ratio of 0.1767. MDP reports explanations in the form of combinations of attributes, each subset of which has risk ratio and support above threshold.

5.2 Basic Explanation Strategy

A naïve solution to computing the risk ratio for various attribute sets is to search twice, once over all inlier points and once over all outlier points, and then look for differences between the inlier and outlier sets. As we experimentally demonstrate in Section 7, this is inefficient as it wastes time searching over attributes in inliers that are eventually filtered due to insufficient outlier support. Moreover, the number of outliers is much smaller than the inliers, so processing the two sets independently ignores the possibility of additional pruning. To reduce this wasted effort, MacroBase takes advantage of both the cardinality imbalance between inliers and outliers as well as the joint explanation of each set.

Optimization: Exploit Cardinality Imbalance. The cardinality of the outlier set is by definition much smaller than that of the inlier set. Therefore, instead of searching the outlier supports and the inlier supports separately, MDP first finds outlier attribute sets with minimum support and subsequently searches the inlier attributes, while only searching for attributes that were supported in the outliers. This reduces the space of inlier attributes to explore.

Optimization: Individual Item Ratios are Cheap. We have found that many important attribute combinations (i.e., with high risk ratio) can be explained by a small number of attributes (typically, one or two, which can be tested inexpensively). Moreover, while computing risk ratios for all attribute combinations is expensive (combinatorial), computing risk ratios for single attributes is inexpensive: we can compute support counts over both inliers and outliers via a single pass over the attributes. Accordingly, MDP first computes risk ratios for single attribute values, then computes support of combinations whose members have sufficient risk ratios.

In contrast with [74], this optimization for risk ratio computation is enabled by the fact that we wish to find *combinations* of attributes whose subsets are each supported and have minimum risk ratio. If a set of attributes is correlated, reporting them as a group helps avoid overwhelming the user with explanations.

Algorithms and Data Structures. In the one-pass batch setting, single attribute value counting is straightforward, requiring a single pass over the data; the streaming setting below is more interesting. We experimented with several itemset mining techniques that use dynamic programming to prune the search over attribute combinations with sufficient support and ultimately decided on prefix-tree-based approaches inspired by FPGrowth [58]. In brief, the FPGrowth algorithm maintains a frequency-descending prefix tree of attributes that can subsequently be mined by recursively generating a set of "conditional" trees. Corroborating recent benchmarks [49], the FPGrowth algorithm was fast and proved extensible in our streaming implementation below.

End Result. The result is a three-stage process (Algorithm 2). MDP first calculates the attribute values with minimum risk ratio (via support counting, followed by a filtering pass based on risk ratio). From the first stage's outlier attribute values, MDP then computes supported outlier attribute combinations using the FPGrowth algorithm. Finally, MDP computes the risk ratio for each attribute combination based on their support in the inliers (via support counting, followed by a filtering pass to exclude any attribute combinations with insufficient risk ratio).

5.3 Streaming Explanation

As in MDP detection, streaming explanation generation is more challenging. We present the MDP implementation of single-attribute streaming explanation, then extend the approach to multi-attribute streaming explanation.

Implementation: Single Attribute Explanation. To begin, we find individual attributes with sufficient support and risk ratio while both respecting changes in the stream and limiting the memory required to store support counts. The problem of maintaining a count of frequent items (i.e., *heavy hitters*, or attributes with top k occurrence) in data streams is well studied [36]. Given a heavy-hitters sketch over the inlier and outlier stream, we can compute an approximate support and risk ratio for each attribute by comparing the contents of the sketches at any time.

Initially, we implemented the MDP item counter using the SpaceSaving algorithm [81], which provides empirically good performance [37] and has extensions in the exponentially decayed setting [38]. However, like many of the sketches in the literature, SpaceSaving was designed to strike a balance between sketch size and performance, with a strong emphasis on limited size. For example, in its heap-based variant, SpaceSaving maintains $\frac{1}{k}$-approximate counts for the top k item counts by maintaining a heap of the items. For a stream of size n, this requires $O(n \log(k))$ update time. (In the case of exponential decay, the linked-list variant can require $O(n^2)$ processing time.)

While logarithmic update time is modest for small sketches, given only two heavy-hitters sketches per MacroBase query, MDP can expend more memory on its sketches to improve accuracy; for example, 1M items require four megabytes of memory for float-encoded counts, which

ALGORITHM 3: AMC: Amortized Maintenance Counter

```
given $\epsilon \in (0,1)$; $r$: decay rate $\in (0,1)$;
initialization $C$ (item $\to$ count) $\leftarrow \{\}$; weight $w_i \leftarrow 0$;
observe($i$: item, $c$: count)
    $C[i] \leftarrow w_i + c$ if $i \notin C$ else $C[i] + c$;
maintain()
    remove all but the $\frac{1}{\epsilon}$ largest entries from $C$;
    $w_i \leftarrow$ the largest value just removed, or, if none removed, 0;
decay()
    decay the value of all entries of $C$ by $r$;
    call maintain();
```

is small relative to modern server memory sizes. As a result, we developed a heavy-hitters sketch, called the *Amortized Maintenance Counter* (AMC, Algorithm 3), that occupies the opposite end of the design spectrum: the AMC uses a much greater amount of memory for a given accuracy level, but is faster to update and still limits total space utilization. The key insight behind the AMC is that if we observe even a single item in the stream more than once, we can amortize the overhead of maintaining the sketch across multiple observations of the same item. In contrast, SpaceSaving maintains the sketch for every observation but in turn ensures a smaller sketch size.

AMC provides the same counting functionality as a traditional heavy-hitters sketch but exposes a second method, *maintain*, that is called to periodically prune the sketch size. AMC allows the sketch size to increase between calls to *maintain*, and, during maintenance, the sketch size is reduced to a desired stable size, which is specified as an input parameter. Therefore, the maximum size of the sketch is controlled by the period between calls to *maintain*: as in SpaceSaving, a stable size of $\frac{1}{\epsilon}$ yields an $n\epsilon$ approximation of the count of n points, but the size of the sketch may grow within a period. This separation of insertion and maintenance has two implications. First, it allows constant-time insertion, which we describe below. Second, it allows a range of maintenance policies, including a sized-based policy, which performs maintenance once the sketch reaches a pre-specified upper bound, as well as a variable period policy, which operates over real-time and tuple-based windows (similar to ADR).

To implement this functionality, AMC maintains a set of approximate counts for all items that were among the most common in the previous period along with approximate counts for all other items that were observed in the current period. During maintenance, AMC prunes all but the $\frac{1}{\epsilon}$ items with highest counts and records the maximum count that is discarded (w_i). Upon insertion, AMC checks to see if the item is already stored. If so, the item's count is incremented. If not, AMC stores the item count plus w_i. If an item is not stored in the current window, the item must have had count less than or equal to w_i at the end of the previous period.

AMC has three major differences compared to SpaceSaving. First, AMC updates are constant time (hash table insertion) compared to $O(\log(\frac{1}{\epsilon}))$ for SpaceSaving. Second, AMC has an additional maintenance step, which is amortized across all items seen in a window. Using a min-heap, with I items in the sketch, maintenance requires $O(I \cdot \log(\frac{1}{\epsilon}))$ time. If we observe even one item more than once, this is faster than performing maintenance on every observation. Third, AMC has higher space overhead; in the limit, it must maintain all items it has seen between maintenance intervals.

Implementation: Exponentially Damped Streaming Combinations. While AMC tracks single items, MDP also needs to track combinations of attributes. As such, we sought a tree-based technique that would admit exponentially damped arbitrary windows but eliminate the requirement that each attribute be stored in the tree, as in recent proposals such as the CPS-tree [106]. As a result, MDP adapts a combination of two data structures: AMC for the frequent attributes, and an adaptation of the CPS-tree data structure to store frequent attribute combinations. We present algorithms for maintaining the adapted CPS-tree below.

Given the set of recently frequent items, MDP monitors the attribute stream for frequent attribute combinations by maintaining an approximately frequency-descending prefix tree of attribute values: the CPS-tree data structure [106], with several modifications, which we call the M-CPS-tree.

Like the CPS-tree, the M-CPS-tree maintains the basic FP-tree data structures as well as a set of leaf nodes in the tree. However, in an exponentially damped model, the CPS-tree stores at least one node for every item observed in the stream. This is infeasible at scale. As a compromise, the M-CPS-tree only stores items that were frequent in the previous window: at each window boundary, MacroBase updates the frequent item counts in the M-CPS-tree based on its AMC sketch. Items that were frequent in the previous window but were not frequent in this window are removed from the tree. MacroBase then decays all frequency counts in the M-CPS-tree nodes and re-sorts the M-CPS-tree in frequency descending order (as in the CPS-tree, by traversing each path from leaf to root and re-inserting as needed). Subsequently, attribute insertion can continue as in the FP-tree.

Summary. MDP's streaming explanation operator consists of two primary parts: maintenance and querying. When a new data point arrives at the explanation operator, MacroBase inserts each of the point's attributes into an AMC sketch. MacroBase then inserts a subset of the point's attributes into a prefix tree that maintains an approximate, frequency descending order. When a window has elapsed, MacroBase decays the counts of the items and the counts in each node of the prefix tree. MacroBase removes any attributes that are no longer above the support threshold and rearranges the prefix tree in frequency-descending order. To produce explanations, MacroBase runs FPGrowth on the prefix tree.

5.4 Confidence Intervals

To provide confidence intervals on its output explanations and prevent false discoveries (type I errors, our focus here), MDP leverages existing results from the epidemiology literature, applied to the MDP data structures. For a given attribute combination appearing a_o times in the outliers and a_i times in the inliers, with a risk ratio of o, b_o other outlier points, and b_i other inlier points, we can compute a $1 - p\%$ confidence interval as

$$o \pm \exp\left(z_p\sqrt{\frac{1}{a_o} - \frac{1}{a_o + a_i} + \frac{1}{b_o} - \frac{1}{b_o + b_i}}\right),$$

where z_p is the z-score corresponding to the $1 - \frac{p}{2}$ percentile [82]. For example, an attribute combination with risk ratio of 5 that appears in 1% of 10M points has a 95th percentile confidence interval of $(3.93, 6.07)$ (99th: $(3.91, 6.09)$). Given a risk ratio threshold of 3, MacroBase can return this explanation with confidence.

However, because MDP performs a repeated set of statistical tests to find attribute combinations with sufficient risk ratio, MDP is subject to the multiple testing problem: large numbers of statistical tests are statistically likely to contain false positives. To address this problem, MDP can apply a correction to its intervals. For example, under the Bonferroni correction [96], if a user seeks a confidence of $1 - p$ and MDP tests k attribute combinations, MDP should instead assess the confidence for z_p at $1 - \frac{p}{k}$. We can compute k at explanation time by recording the number of support computations.

Table 2. Speedups of Hand-Optimized C++ over Java MacroBase Prototype for Simple Queries (Section 7)

	LS	TS	ES	AS	FS	MS
Throughput (pts/sec)	7.86M	8.70M	9.35M	12.31M	7.05M	6.22M
Speedup over Java	7.46×	24.11×	5.24×	16.87×	5.32×	17.54×

k is likely to be large as, in the limit, MDP may examine the power set of all attribute values in the outliers. However, with fast data, this is less problematic. First, the pruning power of MDP's explanation routine eliminates many tests, thus reducing type I errors. Second, empirically, many of MacroBase's explanations have very high risk ratio—often in the tens or hundreds. This is because many problematic behaviors are highly systemic, meaning large intervals may still be above the user-specified risk ratio threshold. Third, and perhaps most importantly, MacroBase analyzes large streams. In the above example, even with $k = 10M$, the 95th percentile confidence interval is still $(3.80, 6.20)$. Compared to medical studies with study sizes in the range of hundreds of samples, the large volume of data mitigates many of the problems associated with multiple testing. For example, the same $k = 10M$ yields a 95th percentile confidence interval of $(0, 106M)$ when applied to a dataset of only 1,000 points, which is effectively meaningless. (This trend also applies to alternative corrective methods such as the Benjamini-Hochberg procedure [19].) Thus, while the volumes of fast data streams pose significant computational challenges, they can actually improve the statistical quality of analytics results.

6 IMPLEMENTATION

In this section, we describe the MacroBase prototype implementation and runtime. As of February 2017, MacroBase's core comprises approximately 9,400 lines of Java, over 7,000 of which are devoted to operator implementation, along with an additional 1,000 lines of JavaScript and HTML for the front-end and 7,600 lines of Java for diagnostics and prototype pipelines.

We chose Java due to its high productivity, support for higher-order functions, and popularity in open source. However, there is considerable performance overhead associated with the Java virtual machine (JVM). Despite interest in bytecode generation from high-level languages such as Scala and .NET [29, 73], we are unaware of any generally available, production-strength operator generation tools for the JVM. As a result, MacroBase leaves performance on the table in exchange for programmer productivity. To understand the performance gap, we rewrote a simplified MDP pipeline in hand-optimized C++. As Table 2 shows, we measure an average throughput gap of 12.76× for simple queries. JVM code generation will reduce this gap.

MacroBase executes operator pipelines via a custom single-core dataflow execution engine. MacroBase's streaming dataflow decouples producers and consumers: each operator writes (i.e,. pushes) to an output stream but consumes tuples as they are pushed to the operator by the runtime (i.e., implements a `consume(OrderedList<Point>)` interface). This facilitates a range scheduling policies: operator execution can proceed sequentially, or by passing batches of tuples between operators. MacroBase supports several styles of pipeline construction, including a fluent, chained operator API. By default, MacroBase amortizes calls to `consume` across several thousand points, reducing function call overhead. This API also allows stream multiplexing and is compatible with a variety of existing dataflow execution engines, including Storm, Heron, and Amazon Streams, which could act as future execution substrates. We demonstrate interoperability with several existing data mining frameworks in Section 7.5.

The MacroBase prototype does not currently implement fault tolerance, although classic techniques such as state-based checkpointing are applicable here [68], especially as MDP's operators contain modest state. The MacroBase prototype is also oriented toward single-core deployment. For parallelism, MacroBase currently runs one query per core (e.g., one query pipeline per application cluster in a datacenter). We report on preliminary multi-core scale-out results in Section 7.5.

The MacroBase prototype and all code evaluated in this article are available online under a permissive open source license.

7 EVALUATION

In this section, we evaluate the accuracy, efficiency, and flexibility of MacroBase and the MDP operators. We wish to demonstrate the following:

— MacroBase is accurate: on controlled, synthetic data, under changes in stream behavior and over real-world workloads from the literature and in production (Section 7.1).
— MacroBase processes up to 2M points per second per query on a range of real-world datasets (Section 7.2).
— MacroBase's cardinality-aware explanation strategy produces meaningful speedups (average: 3.2× speedup; Section 7.3).
— MacroBase's use of AMC is up to 500× faster than existing sketches on production data (Section 7.3).
— MacroBase's architecture is extensible, which we illustrate via three case studies (Section 7.4).

Experimental Environment. We report results from deploying the MacroBase prototype on a server with four Intel Xeon E5-4657L 2.40GHz CPUs containing 12 cores per CPU and 1TB of RAM. To isolate the effects of pipeline processing, we exclude loading time from our results. By default, we issue MDP queries with a minimum support of 0.1% and minimum risk ratio of 3, a target outlier percentile of 1%, ADR and AMC sizes of 10K, a decay rate of 0.01 every 100K points, retraining each every 100K points, and report the average of at least three runs per experiment. We vary these parameters in subsequent experiments in this section.

Implementation. We describe MacroBase's implementation, dataflow runtime, and approach to parallelism in Section 6.

Large-Scale Datasets. To compare the efficiency of MacroBase and related techniques, we compiled a set of large-scale real-world datasets (Table 4) for evaluation. CMT contains user drives at CMT, including anonymized metadata such as phone model, drive length, and battery drain; Telecom contains aggregate internet, SMS, and telephone activity for a Milanese telecom; Accidents contains United Kingdom road accident statistics between 2012 and 2014, including road conditions, accident severity, and number of fatalities; Campaign contains all US Presidential campaign expenditures in election years between 2008 and 2016, including contributor name, occupation, and amount; Disburse contains all US House and Senate candidate disbursements in election years from 2010 through 2016, including candidate name, amount, and recipient name; and Liquor contains sales at liquor stores across the state of Iowa. All but CMT are (i) publicly accessible, allowing reproducibility, and (ii) representative of many challenges we have encountered in analyzing production data beyond CMT in both scale and behaviors. While none of these datasets contain ground-truth labels, we have verified several of the explanations from our queries over CMT.

7.1 Result Quality

In this section, we focus on MacroBase's statistical result quality. We evaluate precision/recall on synthetic and real-world data and report on experiences from production usage.

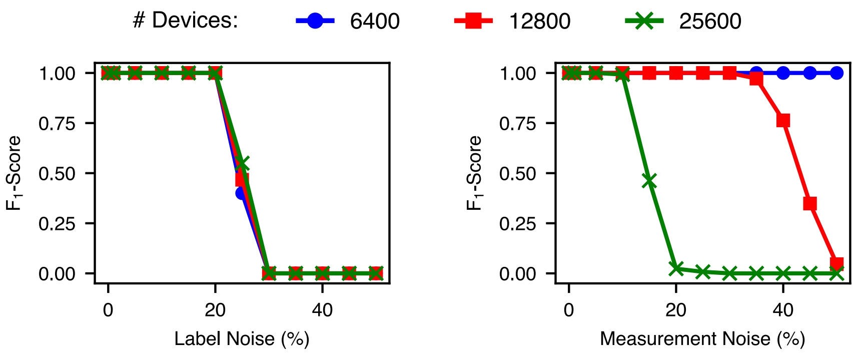


Fig. 4. Precision-recall of explanations. Without noise, MDP exactly identifies misbehaving devices. MDP's use of risk ratio improves resiliency to both label and measure noise.

Synthetic Dataset Accuracy. We ran MDP over synthetic datasets generated in line with those used to evaluate recent anomaly detection systems [94, 112]. The datasets contain 1M data points from 6,400, 12,800, and 25,600 synthetic devices. Each device has a unique device ID attribute and metrics which are drawn from either an inlier distribution ($\mathcal{N}(10, 10)$) or outlier distribution ($\mathcal{N}(70, 10)$). We subsequently evaluated MacroBase's ability to automatically determine the device IDs corresponding to the outlying distribution. We report the F_1-score ($2 \cdot \frac{precision \cdot recall}{precision+recall}$) for the set of device IDs identified as outliers metric for explanation quality.

Since MDP's statistical techniques are a natural match for this experimental setup, we also perturbed the base experiment to understand when MDP might underperform. We introduced two types of noise into the measurements to quantify their effects on MDP's performance. First, we introduced *label noise* by randomly assigning readings from the outlier distribution to inlying devices and vice versa. Second, we introduced *measurement noise* by randomly assigning a proportion of both outlying and inlying points to a third, uniform distribution over the interval $[0, 80]$.

Figure 4 illustrates the results. In the noiseless regions of Figure 4, MDP correctly identified 100% of the outlying devices. As the outlying devices are solely drawn from the outlier distribution, constructing outlier explanations via the risk ratio enables MacroBase to perfectly recover the outlying device IDs. In contrast, techniques that rely solely on individual outlier classification deliver less accurate results on this workload (cf. [94, 112]). Under label noise, MacroBase robustly identified the outlying devices until approximately 25% noise, which corresponds a $3:1$ ratio of correct to incorrect labels. As our risk ratio threshold is set to 3, exceeding this threshold causes rapid performance degradation. Under measurement noise, accuracy degrades linearly with noise. MDP is more robust to this type of noise when fewer devices are present; its accuracy suffers with a larger number of devices, as each device type is subject to more noisy readings.

In summary, MDP is able to accurately identify correlated causes of outlying data for noise of 20% or more. The noise threshold is improved by both MDP's use of robust methods as well as the use of risk ratio to prune irrelevant summaries. Noise of this magnitude is likely rare in practice, and, if such noise exists, is possibly of another interesting behavior in the data.

Real-World Dataset Accuracy. In addition to synthetic data, we also performed experiments to determine MacroBase's ability to accurately identify systemic abnormalities in real-world data. We evaluated MacroBase's ability to distinguish abnormally behaving OLTP servers within a cluster, as defined according to data and manual labels collected in a recent study [113] to diagnose performance issues within a single host. We performed a set of experiments, each corresponding to a distinct type of performance degradation within MySQL on a particular OLTP workload (TPC-C

Table 3. MDP Accuracy on DBSherlock Workload

TPC-C (QS: one MacroBase query per cluster): top-1: 88.8%, top-3: 88.8%									
	A1	A2	A3	A4	A5	A6	A7	A8	A9
Train top-1 correct (of 9)	9	9	9	9	9	8	9	9	8
Holdout top-1 correct (of 2)	2	2	2	2	2	2	2	2	0
TPC-C (QE: one MacroBase query per anomaly type): top-1: 83.3%, top-3: 100%									
	A1	A2	A3	A4	A5	A6	A7	A8	A9
Train top-1 correct (of 9)	9	9	9	9	9	8	9	9	7
Holdout top-1 correct (of 2)	2	2	2	2	2	1	2	2	0
TPC-E (QS: one MacroBase query per cluster): top-1: 83.3%, top-3: 88.8%									
	A1	A2	A3	A4	A5	A6	A7	A8	A9
Train top-1 correct (of 9)	9	9	9	9	9	8	9	9	0
Holdout top-1 correct (of 2)	2	2	2	2	2	1	2	2	0
TPC-E (QE: one MacroBase query per anomaly type): top-1: 94.4%, top-3: 100%									
	A1	A2	A3	A4	A5	A6	A7	A8	A9
Train top-1 correct (of 9)	9	9	9	9	9	8	9	9	6
Holdout top-1 correct (of 2)	2	2	2	2	2	1	2	2	2

A1: workload spike, A2: I/O stress, A3: DB backup, A4: table restore, A5: CPU stress, A6: flush log/table; A7: network congestion; A8: lock contention; A9: poorly written query. "Poor physical design" (from [113]) is excluded as the labeled anomalous regions did not exhibit significant correlations with any metrics.

and TPC-E). For each experiment, we consider a cluster of 11 servers, where a single server exhibits the degradation. Using over 200 operating systems and database performance counters, we ran MDP to identify the anomalous server.

We ran MDP with two sets of queries. In the former set, QS, MDP executed a query to find abnormal hosts (with hostname attributes) using a single set of 15 metrics identified via feature selection techniques on a holdout of two clusters per experiment (i.e., one query for all anomalies). As Table 3 shows, under QS, MDP achieves top-1 accuracy of 86.1% on the holdout set across all forms of anomalies (top-3: 88.8%). For eight of nine anomalies, MDP's top-1 accuracy is higher: 93.8%. However, for the ninth anomaly, which corresponds to a poorly written query, the metrics correlated with the anomalous behavior are substantially different.

In the second set of experiments, QE, MDP executed a slow-hosts query using a set of metrics for each distinct anomaly type (e.g., network contention), again using a holdout of two clusters per experiment (i.e., one query per anomaly type). In contrast with QS, because QE targets each type of performance degradation with a custom set of metrics, it is able to identify behaviors more reliably, leading to perfect top-3 accuracy.

In addition, we plot the CDF of scores in each of our real-world dataset queries in Figure 5. While many points have high outlier scores, the tail of the distribution (at the 99th percentile) is extreme: a very small proportion of points have outlier scores over 150.

These results show that even in unsupervised settings, MacroBase accurately recovers systemic causes and highlights extreme behaviors by focusing on the tails and with proper feature selection.

Production Results. MacroBase currently operates over a range of production data and external users report the prototype has discovered previously unknown and sometimes serious behaviors in several domains. Here, we report on our experiences deploying MacroBase at CMT, where it identified several previously unknown behaviors. In one case, MacroBase highlighted a small number of users who experienced issues with their trip detection. In another case, MacroBase discovered

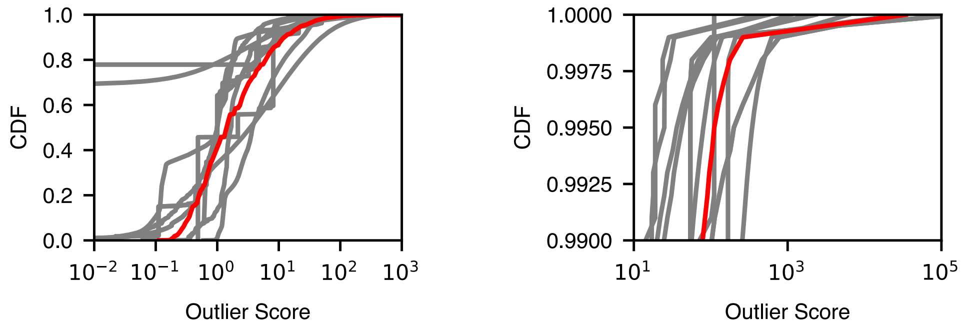


Fig. 5. CDF of outlier scores for all datasets, with average in red; the datasets exhibit a long tail with extreme outlier scores at the 99th percentile and higher.

a rare issue with the CMT application and a device-specific battery problem. Consultation and investigation with the CMT team confirmed these issues as previously unknown, and have since been addressed. These experiences and others [17] have proven a useful demonstration of MacroBase's ability to prioritize attention in production environments and inspired several ongoing extensions (Section 11).

7.2 End-to-End Performance

In this section, we evaluate MacroBase's end-to-end performance on real-world datasets. For each dataset X, we execute two MacroBase queries: a simple query, with a single attribute and metric (denoted XS), and a complex query, with a larger set of attributes and, when available, multiple metrics (denoted XC). We then report throughput for two system configurations: one-shot batch execution that processes each stage in sequence and exponentially weighted streaming execution (EWS) that processes points continuously. One-shot and EWS have different semantics, as reflected in the explanations they produce. One-shot execution examines the entire dataset at once. EWS prioritizes recent points. Therefore, for datasets with few distinct attribute values (e.g., Accidents contains only nine types of weather conditions), the explanations will have high similarity. However, explanations differ in datasets with many distinct attribute values (typically the complex queries with hundreds of thousands of possible combinations—e.g., Disburse has 138,338 different disbursement recipients). For this reason, we provide throughput results both with and without explanations, as well as the number of explanations generated by the simple (XS) and complex (XC) queries and their Jaccard similarity.

Table 4 displays results across all queries. Throughput varied from 147K points per second (on MC with explanation) to over 2.5M points per second (on TS without explanation); the average throughput for one-shot execution was 1.39M points per second, and the average throughput for EWS was 599K points per second. The better-performing mode depended heavily on the particular dataset and characteristics. In general, queries with multiple metrics were slower in one-shot than queries with single metrics (due to increased training time, as streaming trains over samples), and EWS typically returned fewer explanations due to its temporal bias. Generating each explanation at the end of the query incurred an approximately 22% overhead. In all cases, these queries far exceed the current arrival rate of data for each dataset. In practice, users tune their decay on a per-application basis (e.g., at CMT, streaming queries may prioritize trips from the last hour to catch errors arising from the most recent deployment). These throughputs exceed those of related techniques we have encountered in the literature (by up to three orders of magnitude); we examine specific factors that contribute to this performance in the next section.

Table 4. Datasets and Query Names, Throughput, and Explanations Produced under One-shot and Exponentially Weighted Streaming (EWS) Execution

Queries					Thru w/o Explain (pts/s)		Thru w/ Explain (pts/s)		# Explanations		Jaccard
Dataset	Name	Metrics	Attrs	Points	One-shot	EWS	One-shot	EWS	One-shot	EWS	Similarity
Liquor	LS	1	1	3.05M	1549.7K	967.6K	1053.3K	966.5K	28	33	0.74
	LC	2	4		385.9K	504.5K	270.3K	500.9K	500	334	0.35
Telecom	TS	1	1	10M	2317.9K	698.5K	360.7K	698.0K	469	1	0.00
	TC	5	2		208.2K	380.9K	178.3K	380.8K	675	1	0.00
Campaign	ES	1	1	10M	2579.0K	778.8K	1784.6K	778.6K	2	2	0.67
	EC	1	5		2426.9K	252.5K	618.5K	252.1K	22	19	0.17
Accidents	AS	1	1	430K	998.1K	786.0K	729.8K	784.3K	2	2	1.00
	AC	3	3		349.9K	417.8K	259.0K	413.4K	25	20	0.55
Disburse	FS	1	1	3.48M	1879.6K	1209.9K	1325.8K	1207.8K	41	38	0.84
	FC	1	6		1843.4K	346.7K	565.3K	344.9K	1710	153	0.05
CMT	MS	1	1	10M	1958.6K	564.7K	354.7K	562.6K	46	53	0.63
	MC	7	6		182.6K	278.3K	147.9K	278.1K	255	98	0.29

MacroBase sustains throughput of several hundred thousand (and up to 2.5M) points per second.

Runtime Breakdown. To further understand how each pipeline operator contributed to overall performance, we profiled MacroBase's one-shot execution (EWS was challenging to instrument accurately due to its streaming execution). On *MC*, MacroBase spent approximately 52% of its execution training MCD, 21% scoring points, and 26% generating explanations. On *MS*, MacroBase spent approximately 54% of its execution training MAD, 16% scoring points, and 29% generating explanations. In contrast, on *FC*, which returned over 1,000 explanations, MacroBase spent 31% of its execution training MAD, 4% scoring points, and 65% generating explanations. Thus, the overhead of each component is data- and query-dependent.

7.3 Microbenchmarks and Comparison

In this section, we explore three key aspects of MacroBase's design: cardinality-aware explanation, use of AMC sketches, and ADR samplers.

Cardinality-Aware Explanation. We evaluated the efficiency of MacroBase's cardinality-aware pruning versus traditional FPGrowth. MacroBase leverages a unique pruning strategy that exploits the low cardinality of outliers, which delivers large speedups—on average, over 3× compared to unoptimized FPGrowth. Specifically, MacroBase's produces a summary of each dataset's inliers and outliers in 0.22–1.4 seconds. In contrast, running FPGrowth separately on inliers and outliers was, on average, 3.2× slower; compared to MacroBase's joint explanation according to support and risk ratio, much of the time spent mining inliers (with insufficient risk ratio) in FPGrowth is wasted.

However, both MacroBase and FPGrowth must perform a linear pass over all of the inliers, which places a lower bound on the running time. The benefit of this pruning strategy depends on the support and risk ratio threshold on explanation. We varied each and measured the resulting runtime and the number of explanations produced on the EC and MC datasets, which we plot in Figure 6. Each dataset has few attributes with outlier support greater than 10%, but each had over 1,700 with support greater than 0.001%. Modifying the support threshold beyond 0.01% had limited impact on runtime; most time in explanation is spent in simply iterating over the inliers rather than maintaining tree structures. This effect is further visible when varying the risk ratio,

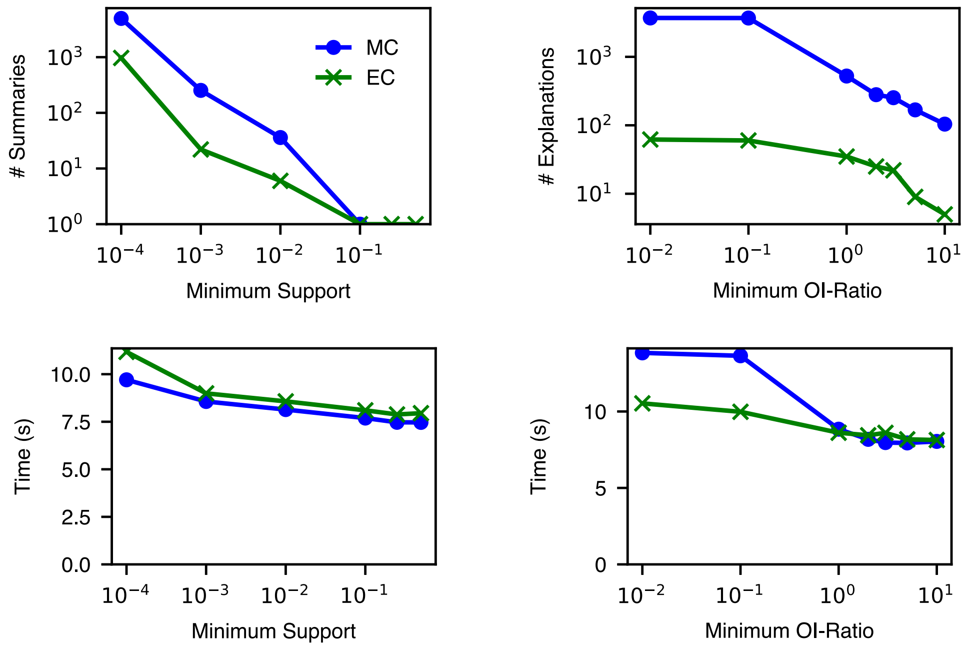


Fig. 6. Explanations produced and explanation time under varying support (percentage) and risk ratio.

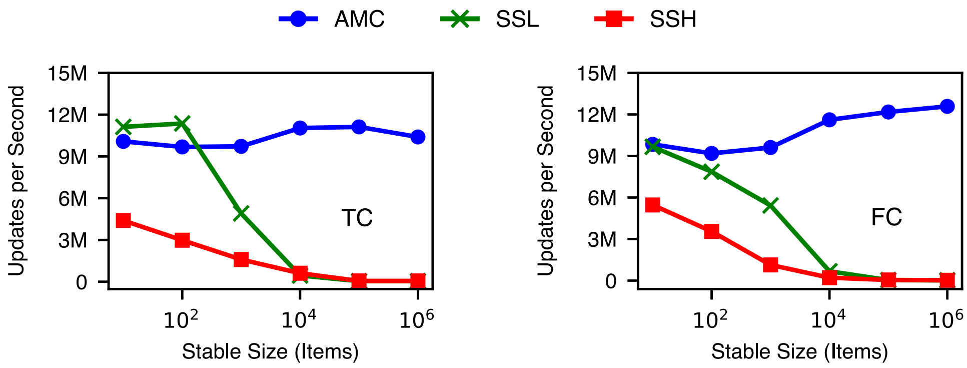


Fig. 7. Streaming heavy hitters sketch comparison. AMC: Amortized Maintenance Counter with maintenance every 10K items; SSL: Space Saving List; SSH: Space Saving Hash. All share the same accuracy bound. Varying the AMC maintenance period produced similar results.

which has less than 40% impact on runtime yet leads to an order of magnitude change in number of explanations. Our default setting of support and risk ratio yields a sensible tradeoff between number of explanations produced and runtime.

AMC Comparison. We also compared the performance of AMC with existing heavy-hitters sketches (Figure 7). AMC outperformed both implementations of SpaceSaving in all configurations by a margin of up to 500× for sketch sizes exceeding 100 items. This is because the SpaceSaving overhead (heap maintenance on every operation is expensive with even modestly sized sketches or list traversal is costly for decayed, non-integer counts) is costly. In contrast, with an update

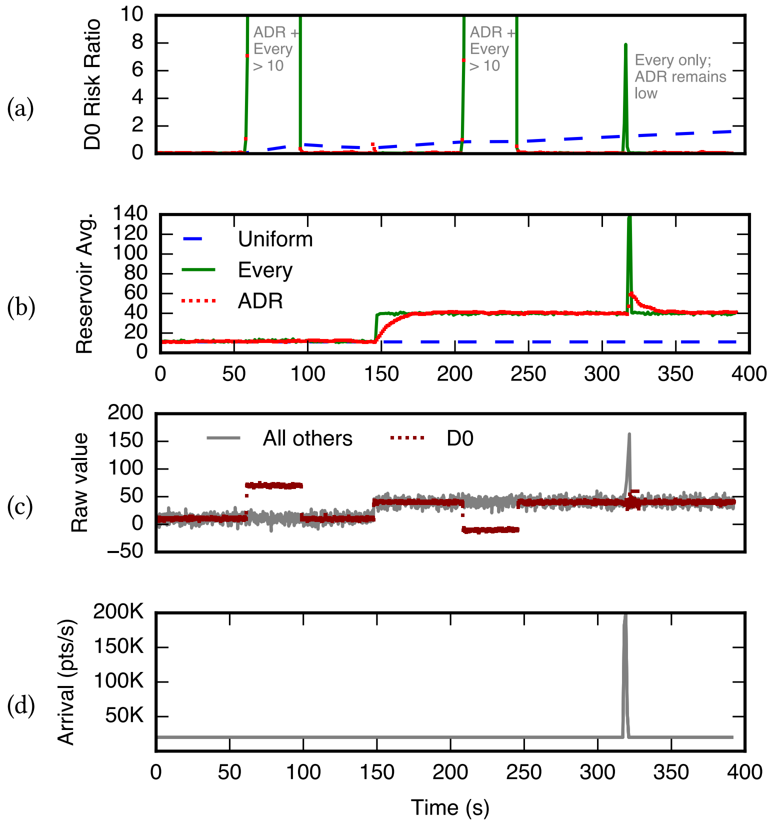


Fig. 8. ADR provides greater adaptivity compared to tuple-at-a-time reservoir sampling and is more resilient to spikes in data volume (see text for details).

period of 10K points, AMC sustained over 10M updates per second. The primary cost of these performance improvements is additional space: for example, with a minimum sketch size of 10 items and update period of 10K points, AMC retained up to 10,010 items while each SpaceSaving sketch retained only 10. As a result, when memory sizes are especially constrained, SpaceSaving may be preferable, at a measurable cost to performance.

ADR Adaptivity. While the previous set of experiments operated over data with a static underlying distribution, we sought to understand the benefit of MDP's ability to adapt to changes in the input distribution via the exponential decay of ADR and AMC. We performed a controlled experiment over two types of time-varying behavior: changing underlying data distribution, and variable data arrival rate. We then compared the accuracy of MDP outlier detection across three sampling techniques: a uniform reservoir sample, a per-tuple exponentially decaying reservoir sample, and our proposed ADR.

Figure 8(c) displays the time-evolving stream representing 100 devices over which MDP operates. To begin, all devices produce readings drawn from a Gaussian $\mathcal{N}(10, 10)$ distribution. After 50 seconds, a single device, $D0$, produces readings from $\mathcal{N}(70, 10)$ before returning to the original distribution at 100 seconds. The second period (150s to 300s) is similar to the first, except we also introduce a shift in all devices' metrics: after 150 seconds, all devices produce readings from $\mathcal{N}(40, 10)$, and, after 225 seconds, $D0$ produces readings from $\mathcal{N}(-10, 10)$, returning to $\mathcal{N}(40, 10)$

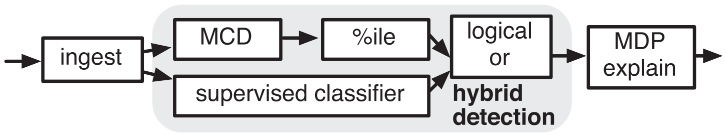
after 250 seconds. Finally from 300s to 400s, all devices experience a spike in data arrival rate. We introduce a 4-second noise spike in the sensor readings at 320 seconds: the arrival rate rises by tenfold, to over 200k points per second, with corresponding values drawn from a $\mathcal{N}(85, 15)$ distribution (Figure 8(d)).

In the first time period, all three strategies detect $D0$ as an outlier, as reflected in the computed risk ratios in Figure 8(a). After 100 seconds, when $D0$ returns to the inlier distribution, its risk ratio drops. The reservoir averages remain unchanged in all strategies (Figure 8(b)). In the second time period, both adaptive reservoirs adjust to the new distribution by 170 seconds, while the uniform reservoir fails to adapt quickly (Figure 8(b)). As such, when $D0$ drops to $\mathcal{N}(-10, 10)$ from time 225 through 250, only the two adaptive strategies track the change (Figure 8(a)). At time 300, the short noise spike appears in the sensor readings. The per-tuple reservoir is forced to absorb this noise, and the distribution in this reservoir spikes precipitously. As a result, $D0$, which remains at $\mathcal{N}(40, 10)$ is falsely suspected as outlying. In contrast, the ADR average value rises slightly but never suspects $D0$ as an outlier. This illustrates the value of MDP's adaptivity to distribution changes and resilience to variable arrival rates.

7.4 Case Studies and Extensibility

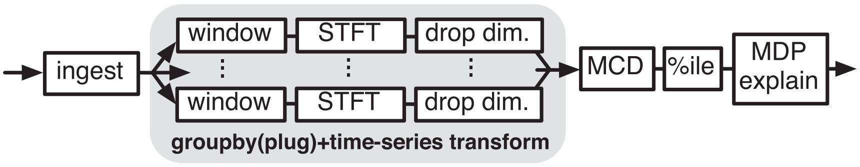
MacroBase is designed for extensibility, as we highlight via case studies in three separate domains. We describe the pipeline structures, performance, and interesting explanations from applying MacroBase over supervised, time-series, and video surveillance data.

Hybrid Supervision. We demonstrate MacroBase's ability to combine supervised and unsupervised classification models via a use case from CMT. Each trip in the CMT dataset is accompanied by a supervised diagnostic score representing the trip quality. While MDP's unsupervised operators can use this score as an input, CMT also wishes to capture low-quality scores independent of their distribution in the population. Accordingly, we authored a new MacroBase pipeline that feeds some metrics (e.g., trip length, battery drain) to the MDP MCD operator and also feeds the diagnostic metric (trip quality score) to a special rule-based operator that flags low-quality scores as anomalies. The pipeline, which we depict below, performs a logical *or* over the two classification results:

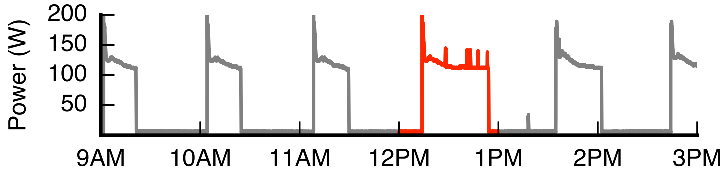


With this hybrid supervision strategy, MacroBase identified additional behaviors within the CMT dataset. Since the quality scores were generated external to MacroBase and the supervision rule in MacroBase was lightweight, runtime was unaffected. This kind of pipeline can easily be extended to more complex supervised models.

Time-Series. MacroBase can also detect temporal behaviors via feature transformation, which we demonstrate using a dataset of 16M points capturing a month of electricity usage from devices within a household [18]. We augment MDP by adding a sequence of feature transforms that (i) partition the stream by device ID, (ii) window the stream into hourly intervals, with attributes according to hour of day, day of week, and date, then (iii) apply a Discrete-Time Short-Term Fourier Transform (STFT) to each window, and truncate the transformed data to a fixed number of dimensions. As the diagram below shows, we feed the transformed stream into an unmodified MDP and search for outlying time periods and devices:

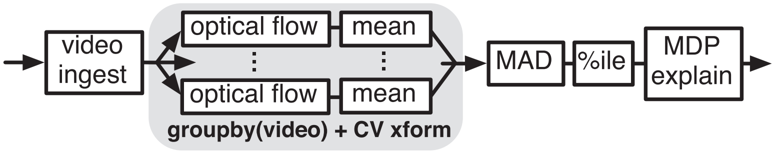


With this custom time-series pipeline, MacroBase detected several systemic periods of abnormal device behavior. For example, the following dataset of power usage by a household refrigerator spiked on an hourly basis (possibly corresponding to compressor activity); instead of highlighting the hourly power spikes, MacroBase was able to detect that the refrigerator consistently behaved abnormally compared to other devices in the household and to other time periods between the hours of 12PM and 1PM—presumably, lunchtime—as highlighted in the excerpt below:

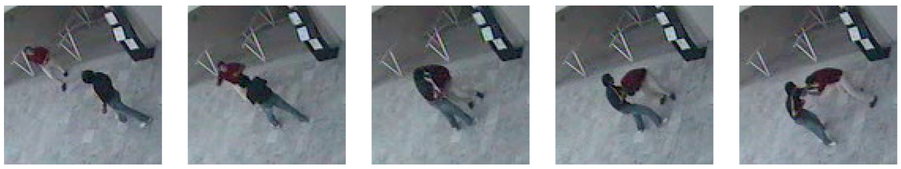


Without feature transformation, the entire MDP pipeline completed in 158ms. Feature transformation on the 16M points dominated the runtime, utilizing 516 seconds via unoptimized STFT.

Video Surveillance. We further highlight MacroBase's ability to easily operate over a wide array of data sources and domains by searching for interesting patterns in the CAVIAR video surveillance dataset [3]. Using OpenCV 3.1.0, we add a custom feature transform that computes the average optical flow velocity between video frames, a technique that has been successfully applied in human action detection [42]. Each transformed frame is tagged with a time interval attribute, which we use to identify interesting video segments and, as depicted below, the remainder of the pipeline executes the standard MDP operators:



Using this pipeline, MacroBase detected periods of abnormal motion in the video dataset. For example, the MacroBase pipeline highlighted a 3-second period in which two people fought:



Like our STFT pipeline, feature transformation via optical flow dominated runtime (22s vs. 34ms for MDP); this is unsurprising given our CPU-based implementation of an expensive transform but nevertheless illustrates MDP's ability to process video streams.

7.5 Additional Evaluation

In addition to the above key macro- and micro-benchmarks, we provide additional results examining the distribution of outlier scores, the effect of varying support and risk ratio, the effect of

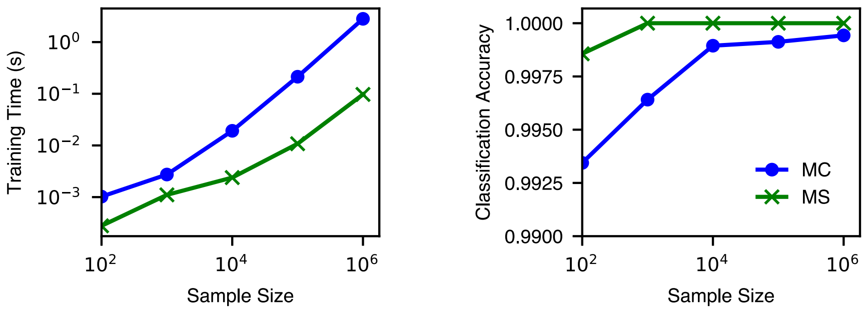


Fig. 9. Behavior of MAD (MS) and MCD (MC) on samples.

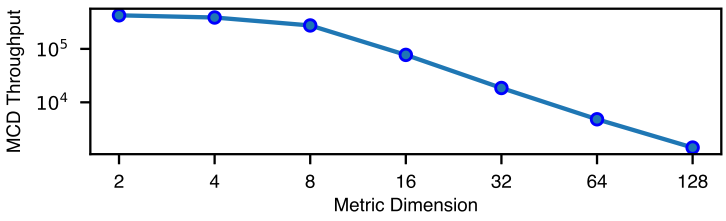


Fig. 10. MCD throughput versus metric size.

training over samples and operating over varying metric dimensions, the behavior of the M-CPS tree, preliminary scale-out behavior, comparing the runtime of MDP explanation to both existing batch explanation procedures, and MDP detection and explanation to operators from frameworks including Weka, Elki, and RapidMiner.

Classification: Retraining on Reservoir Samples. MDP periodically trains models using samples from the input distribution. The statistics literature offers confidence intervals on the MAD [101] and the Mahalanobis distance [89] (e.g., for a sample of size n, the confidence interval of MAD shrinks with $n^{1/2}$), while MCD converges at a rate of $n^{-1/2}$ [24]. To empirically evaluate these effects, we measured the accuracy and efficiency of training models on samples from a $10M$ point dataset. In Figure 9, we plot the outlier classification accuracy versus sample size for the CMT queries. MAD precision and recall are largely unaffected by sampling, allowing a two order-of-magnitude speedup without loss in accuracy. In contrast, MCD accuracy is slightly more sensitive due to variance in the sample selection. This variance is partially offset by the fact that models are retrained regularly under streaming execution, and the resulting speedups in both models are substantial.

Classification: Metric Scalability. As Figure 10 demonstrates, MCD train and score throughput (here, over Gaussian data) is linearly affected by data dimensionality, encouraging the use of dimensionality reduction techniques for complex data.

Explanation: M-CPS and CPS Behavior. We also investigated the behavior of the M-CPS-tree compared to the generic CPS-tree. The two data structures have different behaviors and semantics: the M-CPS-tree captures only itemsets that are frequent for at least two windows by leveraging an AMC sketch. In contrast, CPS-tree captures all frequent combinations of attributes but must insert each point's attributes into the tree (whether supported or not) and, in the limit, stores (and re-sorts) all items ever observed in the stream. As a result, across all queries except ES and EC, the CPS-tree was on average 130× slower than the M-CPS-tree (std. dev.: 213×); on ES and EC,

Table 5. Running Time of Explanation Algorithms (s) for Each Complex Query

Query	MB	FP	Cube	DT10	DT100	AP	XR
LC	1.01	4.64	DNF	7.21	77.00	DNF	DNF
TC	0.52	1.38	4.99	10.70	100.33	135.36	DNF
EC	0.95	2.82	16.63	16.19	145.75	50.08	DNF
AC	0.22	0.61	1.10	1.22	1.39	9.31	6.28
FC	1.40	3.96	71.82	15.11	126.31	76.54	DNF
MC	1.11	3.23	DNF	11.45	94.76	DNF	DNF

MB: MacroBase, FP: FPGrowth, Cube: Data cubing; DT*X*: decision tree, maximum depth *X*; AP: A-Aprioi; XR: Data X-Ray. DNF: did not complete in 20 minutes.

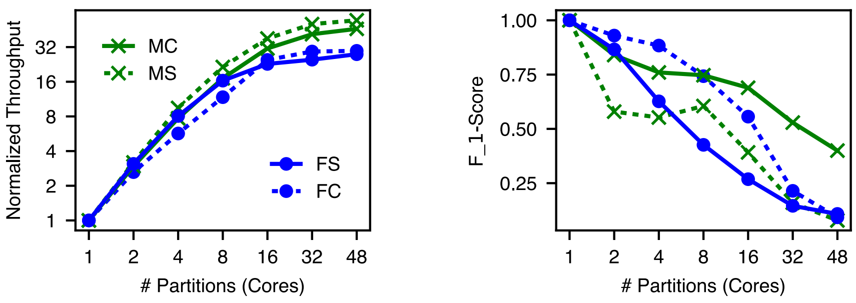


Fig. 11. Behavior of naive, shared-nothing scale-out.

the CPS-tree was over 1,000× slower. The exact speedup was influenced by the number of distinct attribute values in the dataset: Accidents had few values, incurring 1.3× and 1.7× slowdowns, while Campaign had many, incurring substantially greater slowdowns

Explanation: Runtime Comparison. Following the large number of recent data explanation techniques (Section 10), we implemented several additional methods. The results of these methods are not comparable, and prior work has not evaluated these techniques with respect to one another in terms of semantics or performance. We do not attempt a full comparison based on semantics but do perform a comparison based on running time, which we depict in Table 5. We compared to a data cubing strategy suggested by Roy and Suciu [95], which generates counts for all possible combinations (21× slower), Apriori itemset mining [57] (over 43± slower), and Data X-Ray [109]. Cubing works better for data with fewer attributes, while Data X-Ray is optimized for hierarchical data; we have verified with the authors of Data-X-Ray that, for MacroBase's flat attributes, Data-X-Ray will consider all combinations unless stopping criteria are met. MacroBase's cardinality-aware explanation completes fastest for all queries.[5]

Preliminary Scale-Out. As a preliminary assessment of MacroBase's potential for scale-out, we examined MDP behavior under a naive, shared-nothing parallel execution strategy. We partitioned the data across a variable number of cores of a server containing four Intel Xeon E7-4830 2.13 GHz CPUs and processed each partition in parallel; upon completion, we return the union of each core's explanation. Figure 11 shows this strategy delivers excellent linear scalability. However, as each core processes a sample of the overall dataset, accuracy suffers due to both model drift (as in

[5] As noted in Section 7.2, the Accidents data has few distinct attribute values. Therefore, query AC's runtime is bottlenecked not on explanations, but on data scans, as reflected in the near identical performance between DT10 and DT100.

Figure 9) and lack of cross-partition cooperation in explanation. For example, with 32 partitions spanning 32 cores, FS achieves throughput nearing 29M points per second with perfect recall, but only 12% accuracy. Improving accuracy while maintaining scalability is an area of ongoing work.

Compatibility with Existing Frameworks. We implemented several additional MacroBase operators to validate interoperability with existing data mining packages. We were unable to find a single framework that implemented both unsupervised outlier detection and data explanation and had difficulty locating streaming implementations. Nevertheless, we implemented two MacroBase outlier detection operators using Weka 3.8.0's `KDTree` and Elki 0.7.0's `SmallMemoryKDTree`, an alternative FastMCD operator based on a recent RapidMiner extension (`CMGOSAnomalyDetection`) [54], an alternative MAD operator from the OpenGamma 2.31.0, and an alternative FPGrowth-based summarizer based on SPMF version v.0.99i. As none of these packages allowed streaming operation (e.g., Weka allows adding points to a `KDTree` but does not allow removals, while Elki's `SmallMemoryKDTree` does not allow modification), we implemented batch versions. We do not perform accuracy comparisons here but note that the kNN performance was substantially slower (>100×) than MDP's operators (in line with recent findings [54]) and, while SPMF's operators were faster than our generic FPGrowth implementation, SPMF was still 2.8× slower than MacroBase due to MDP's cardinality-aware optimizations. The primary engineering overheads came from adapting to each framework's data formats; however, with a small number of utility classes, we were able to easily compose operators from different frameworks and also from MacroBase, without modification. Should these frameworks begin to prioritize streaming execution and/or explanation, this interoperability may prove fruitful in the future.

8 BEYOND MACROBASE'S DEFAULT PIPELINE

As a prototype fast data engine, MacroBase is the subject of active research and development. Inspired by several real-world MacroBase use cases, we continue to evolve the engine beyond the MDP pipeline in response to user demand, performance considerations, and usability.

In this section, we describe four key workloads and resulting modifications for operating on data cubes and aggregated data (Section 8.1), performing window-based explanation (Section 8.2), supporting alternative ratio-based metrics in explanation (Section 8.3), and enabling efficient multivariate feature selection via sampling (Section 8.4).

8.1 Operating on Aggregated Data

MacroBase ingest operators work naturally with a variety of powerful data stores including SQL databases and OLAP engines. These ingest sources provide MacroBase a stream of `Points`, each corresponding to a single observation.

However, given access to these relational engines, we can also leverage their ability to efficiently pre-process data via aggregation. Moreover, we have found several users are interested in processing *pre-aggregated* results (i.e., tables where rows have already been grouped by the attributes a user is interested in examining). In practice, these pre-aggregated results are often materialized views of measurements. They have counts and other summary statistics rolled-up along dimensions of interest, and serve as a compact representation of the raw data stream.

When aggregated results are easily accessible, they present opportunities for optimizing the explanation portion of the MDP pipeline, as well as challenges for defining semantics for explanations where the original data is not available.

8.1.1 Explanation over Aggregates. As MDP's classification operators process data on a per-point basis, performing outlier detection on aggregated data is challenging. Thus, we first focus on explanation, and describe how to perform explanation when outliers and inliers are classified and

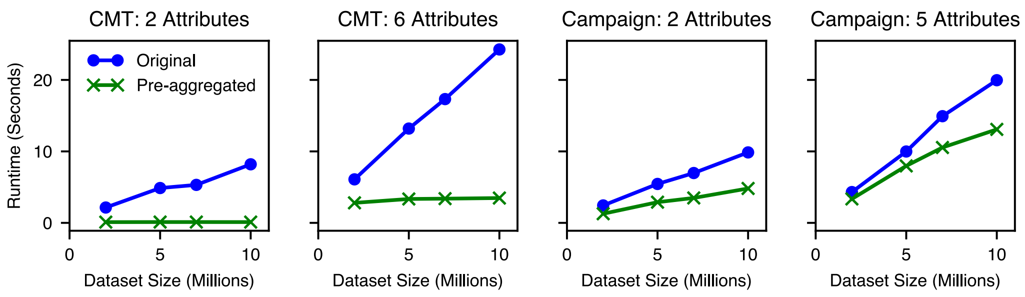


Fig. 12. Time to run explanation operator on pre-aggregated data with varying numbers of attributes. Performance is better when there are fewer attribute columns and a limited number of distinct attribute value combinations.

aggregated separately. In this scenario, outlier classification can be pushed down into the ingest source—this is useful for sophisticated users who apply pre-existing, hand-tuned outlier classification prior to MacroBase ingestion. Hence, the MDP outlier classification step is unnecessary.

Recall that the MDP explanation operator by default ingests individual data points, each of which has a set of associated attributes. Perhaps surprisingly, it is possible to run the MDP explanation operator over aggregated data with only a small modification to the core routines. We begin with a discussion of the batch routines.

If the data is not already aggregated, we first perform the equivalent of a relational group-by on each attribute column to count the total number of outliers and inliers. This gives us a stream of the form `stream<(attrs,label,count)>` describing the number of inliers and outliers for distinct attribute combinations. For example, given attributes `user_id`, `hw_make`, and `hw_model`, we would compute the count of outliers and inliers for each combination of `user_id`, `hw_make`, and `hw_model` appearing in the data.

Second, to perform explanation on this aggregated data, we can execute the explanation algorithm described in Section 5 by treating each distinct attribute-value combination as a weighted `Point`—instead of incrementing inlier and outlier supports on a per-point basis, we can update the support for attribute-value combinations based on the number of points in the group. Data structures such as the FP-Tree and M-CPS-Tree allow arbitrary weighting of points (as used in exponential decay). Here, we weight each group of points by their respective inlier and outlier counts.

Performing explanations on aggregate groups instead of individual points provides the same result with less computation. Specifically, in a dataset with N points and C distinct attribute-value combinations, pre-aggregation reduces the cost of constructing the explanation index from $O(N)$ to $O(C)$. When there are many measurements from a single device or configuration, this represents substantial savings. To illustrate this effect, we evaluate the explanation operator on both CMT and Campaign datasets with 1 metric and a varying number of attributes. Figure 12 illustrates how the runtime of operating over raw data compares with the runtime of explanation on pre-aggregated data. The runtime is nearly constant for the CMT data, because the attribute columns have relatively low cardinality and the number of attribute combinations is smaller. The Campaign dataset contains a more diverse set of attributes corresponding to individual employers and zipcodes and thus exhibits a smaller but still noticeable speedup.

Thus, when the cardinality of attribute-value pairs in a dataset is relatively small, and/or aggregating data using a relational group-by operator is inexpensive (or may already have been performed, as in a data cube), performing explanation on aggregated data can provide substantial runtime reduction with limited additional implementation complexity.

8.1.2 Full Operation over Aggregates. In the previous section, we saw that performing explanation on aggregated data can improve runtime when inlier and outlier classification can be computed before aggregation. However, since MacroBase supports unsupervised classification, we describe alternative semantics that extend the MDP pipeline to operate on pre-aggregated streams without classification labels. These semantics provide the same performance improvement as in Figure 12, and allow MacroBase to operate directly over available streams with arithmetic aggregates.

We are interested here in streams of the form `stream<(attrs,count,average,stddev)>` where the inliers and outliers are grouped together and aggregate statistics such as average (mean) and standard deviation are reported along with a total count. These streams are the common products of simple group-by and cube operations in various relational and OLAP engines.

As noted above, the challenge with these streams is that it is impossible to calculate a risk ratio and support using MAD and MCD without fine-grained statistics. Thus, instead of relying on MAD, MCD, outlier risk ratio, and support, we perform explanation over a subgroup by measuring its distance from the mean and its relative size. Given a dataset with mean μ, variance σ^2, and size n, as well as an attribute combination with corresponding mean μ_i and size n_i, we define $Z_i = \frac{|\mu_i - \mu|}{\sigma}$ and $S_i = \frac{n_i}{n}$. We then run the MDP pipeline with these quantities in place of risk ratio and support using the weighted explanation as above. This produces explanations corresponding to subgroups with mean that deviates substantially from the overall population (i.e., a "mean shift"). Since the semantics here are different, the results will not agree with what MacroBase would have output using a standard MDP pipeline that operates on risk ratio and support. However, we have empirically found that explanations with high risk ratio and support often exhibit high mean shift (i.e., Z_i) as well, thus allowing a reasonable alternative for pre-aggregated datasets that contain only mean, variance, and count.

8.2 Explanation over Sliding Windows

By default, MacroBase's MDP pipeline produces exponentially damped streaming explanations (Section 5.3). However, several users indicated a preference for producing explanations with sliding-window semantics instead of exponentially damped windows; one concern is that it can be difficult for less mathematically inclined users to interpret the semantics of a "decaying" signal. In contrast, sliding-window semantics are common, especially in monitoring scenarios. For example, for monitoring purposes, a user may be interested in obtaining an hourly update of the outlying behaviors in her system over the past 24 hours. This scenario can be cleanly represented in the sliding-window model, as it corresponds to generating explanations over a sliding window with a duration of 24 hours and a slide size of 1 hour.

As a strawman solution, we could simply invoke the batch explanation procedure on the current sliding window whenever the user wants to generate explanations. However, this requires retaining all the data in the window, and incurs unnecessary overhead for sliding windows with significant overlap when users want to generate explanations frequently.

Instead, we incrementally compute explanations on a sub-window basis; following the common practice of efficient evaluating sliding-window aggregates [75], we perform explanation incrementally at the granularity of "panes." Specifically, given window size w and slide size s, the pane size is defined as the greatest common divider of the two parameters ($gcd(w, s)$).

Similar to our exponentially weighted M-CPS-Tree, the challenge in incrementally maintaining pane-based sliding-window explanations is that to provide exact explanations over the entire window, we must track the occurrences of every pattern in the window [106]. As motivation, consider a MacroBase query with minimum support threshold of 1,000 for a window with 100 panes and an attribute-value combination (combo) c that first occurs 999 times in the first pane and occurs

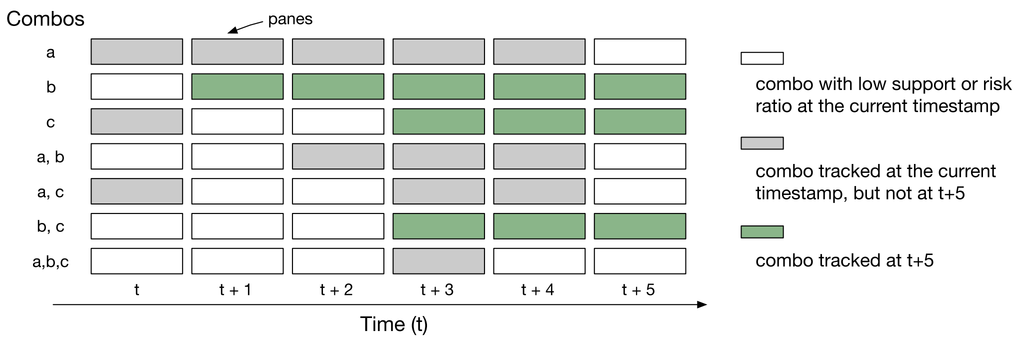

Combos
panes
a
b
c
a, b
a, c
b, c
a,b,c
t
t + 1
t + 2
t + 3
t + 4
t + 5
Time (t)
combo with low support or risk ratio at the current timestamp
combo tracked at the current timestamp, but not at t+5
combo tracked at t+5

Fig. 13. Illustration of the windowed streaming explanation operator semantics. A block of consecutive shaded panes indicates that the corresponding combo has had sufficient support and risk ratio in the time period. For example, combo c has sufficient support and risk ratio at time t, time $t + 3$, time $t + 3$ to $t + 4$, and time $t + 3$ to $t + 5$.

twice in the last pane (without any occurrences in the panes between). For the first 99 panes, c has insufficient support and does not contribute to an explanation. Moreover, if the final pane does not include c, then storing c would not be necessary. For data streams with a large number of distinct combinations, maintaining these sub-threshold counts for each pane is expensive, and can even require storing the entire window contents.

To avoid the overhead of storing every attribute value appearing in a window, we amend the semantics for sliding-window explanations. Instead of reporting all attribute values with sufficient support and risk ratio for the entire window, we report attribute-value combinations with sufficient support risk ratio over a *contiguous suffix* of panes. Figure 13 illustrates these semantics. If an attribute-value explanation has sufficient support and risk ratio for each pane in the window, it will be returned. However, if the attribute-value combination does not have sufficient support or risk ratio for the entire window, it will still be returned provided that it has sufficient support and risk ratio for the last pane, and possibly many panes preceding. In Figure 13, at time $t + 5$, only itemsets $\{b\}, \{c\}, and \{b, c\}$ have continuous support and risk ratio and therefore will be included in the explanation. These semantics allow users to query for explanations at each slide while tracking long-term behavior over the duration of the window.

We present an implementation of the above semantics in Algorithm 4. Upon the arrival of a new data pane, we first count the occurrences of attribute-value combinations that previously passed the support and risk ratio thresholds (i.e., *tracked* combinations); via the `addNewPane()` function. With these pane counts, we check whether all tracked combinations have sufficient support and risk ratio in a contiguous sequence of panes ending with the current pane and remove any combinations that failed the check (`demote()` function). We then run the FPGrowth algorithm to identify new combinations that exceed the support and risk ratio thresholds in the current pane. We mark these combinations as being tracked and start storing their inlier and outlier counts from the current pane (`promote()` function).

The above procedure allows us to track "interesting" explanations that appear in the recent sequences of panes, as opposed to all explanations. Specifically, we record the contents of each pane, and, following pane completion, only require memory proportional to the output size (i.e., number of explanations). As in the exponentially damped setting, the windows act as a time-limited filter on the events, making sure that we do not consume resources tracking events that began in the distant past. This has proven useful in production, especially in memory-constrained multi-tenant settings.

ALGORITHM 4: Windowed Incremental Summarizer

```
variables tracking_map: Map<combo, starting_pane_num>,
          map of tracked combo to the pane number that it first became promoted
variables pane_counts: List<Map<combo, (inlier_count, outlier_count)>>,
          inlier and outlier counts of tracked combos in each pane
process(curr_pane)
    if pane_counts.size() == #panes in window then
        expireLastPane();
    end
    pane_inliers, pane_outliers = split(curr_pane);
    addNewPane(pane_inliers, pane_outliers);
    demote();
    promote(pane_inliers, pane_outliers);

expireLastPane()
    expired_pane = pane_counts.pollFirst();
    for combo ∈ expired_pane do
        starting_pane_num = tracking_map[combo];
        if starting_pane_num > 0 then
            tracking_map[combo] = starting_pane_num − 1;
        end
    end
addNewPane(pane_inliers, pane_outliers)
    curr_pane_count = {};
    for combo ∈ tracking_map do
        curr_pane_count[combo] = (count(pane_inliers, combo), count(pane_outliers, combo));
    end
    pane_counts.add(curr_pane_count);
demote()
    for combo ∈ tracking_map do
        starting_pane_num = tracking_map[combo];
        support = getSupport(combo, pane_counts[starting_pane_num :]);
        risk_ratio = getRiskRatio(combo, pane_counts[starting_pane_num :]);
        if support < min_support or risk_ratio < min_RR then
            tracking_map.remove(combo);
            pane_counts[−1].remove(combo);
        end
    end
promote(pane_inliers, pane_outliers)
    freq_combos = FPGrowth(pane_outliers, min_support);
    for combo ∈ freq_combos do
        if getRiskRatio(combo, pane_counts[−1]) ≥ min_RR then
            tracking_map[combo] = pane_counts.size() - 1;
            pane_counts[−1][combo] = (count(pane_inliers, combo), count(pane_outliers, combo));
        end
    end
```

We compare the performance of the incremental explanation operator to the baseline solution of repeatedly calling FPGrowth on the CMT *MC* query by measuring average per-window query time with a minimum support threshold of 1% and a minimum risk ratio threshold of 3.

We first fix the window size to 1M data points and measure both explanation operators' query time under slide sizes varying from $\frac{1}{80}$ to $\frac{1}{5}$ of the window (Figure 14, left). For smaller slide sizes

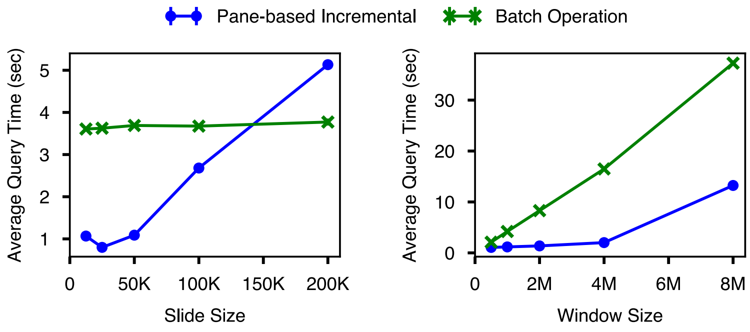


Fig. 14. Incremental and repeated batch explanation performance under varying slide and window sizes. The incremental explanation operator outperforms when users update explanations frequently (small slide size), or when they generate explanations over large historical windows (large window size).

(up to $\frac{1}{10}$ of the window size), the incremental explanation operator outperforms and achieves up to 4.5× decrease in query time compared to the baseline. However, when the slide size accounts for a significant portion of window size (or, when we do not generate explanations frequently), it can be faster to simply call the batch explanation operator on demand (at the cost of additional memory overhead). In addition, the performance of the repeated batch explanation operator is independent of the slide size, as the operator always generates explanations on a fix sized window.

Similarly, we measure the two explanation operators' query time with a fixed slide size of 50K data points and varying window sizes ranging from 10× to 160× of the slide size (Figure 14, right). As opposed to the previous scenario, the incremental explanation operator maintains low query time under varying window sizes, while the repeated batch explanation operator exhibits degraded performance with the increase of window sizes. These experiments suggest that the incremental explanation operator is better suited for low-memory settings and also when users need to frequently update explanations as new data arrives, or when they need to generate explanations over large historical windows.

8.3 Alternative Explanation Metrics

MDP is designed to compute explanations using the relative risk ratio to diagnose factors (i.e., attribute-value combinations) that disproportionately affect a point's likelihood of being considered an outlier. However, in conversations with users, we found there are a number of alternative ratios available in the statistics literature that are also useful in different scenarios.

Among these alternatives, the *rate ratio* has shown to be particularly effective. The rate ratio captures the relative rate of a given attribute-value combination between two different populations—in MDP, the inliers and outliers. Similar to the risk ratio, the rate ratio was developed by epidemiologists [35] to characterize the incidence rate of diseases at different points in time. We have found the rate ratio useful in capturing differences across time (e.g., what attribute-value occurrences most changed between yesterday and today) and in A/B testing, where we wish to compare the rate of occurrence of an outcome (e.g., click-through rate) in two populations.

One key advantage of the rate ratio is that it can be computed from the same underlying counts as the risk ratio: a_o, the number of times a given attribute-value combination appears in the outliers; a_i, the number of times the combination appears in the inliers; b_o, the number of times the combination does not appear in the outliers; and b_i the number of times the combination does not

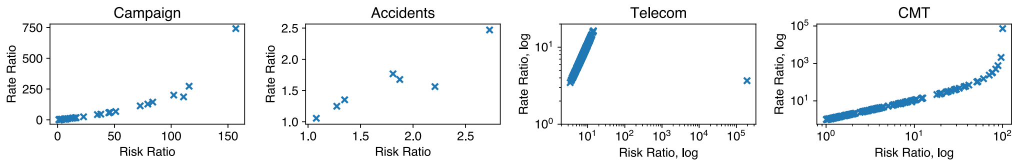


Fig. 15. Distribution of risk ratios and rate ratios for the explanations found in the Campaign, Accidents, Telecom, and CMT datasets. The point on the far right of the Telecom dataset corresponds to an explanation that appears more than 99.99% of the time, thus the risk ratio is significantly higher than the rate ratio.

appear in the inliers. The rate ratio is then

$$\text{rate ratio} = \frac{a_o/(a_o + b_o)}{a_i/(a_i + b_i)}.$$

As the risk ratio and rate ratio both make use of the same counts, we can compute each using the same strategy as used by the default MDP operators.

Given an attribute-value combination c, the rate ratio compares the prevalence of c in the outliers versus the inliers (i.e., $\frac{P(c|out)}{P(c|\neg out)}$). A rate ratio of 3, for example, indicates that c appears three times more often in the outlier population compared to the inlier population. In contrast, the risk ratio compares the likelihood of outliers in points matching c versus points that do not match c (i.e., $\frac{P(out|c)}{P(out|\neg c)}$). Hence, a risk ratio of 3 indicates that points containing c are three times more likely to appear in the outliers than points that do not contain c. In scenarios where outlier occurrence is independent of c, (i.e., $P(out|c) = P(out)$ and $\frac{a_o}{a_i} = \frac{b_o}{b_i}$), the risk ratio and rate ratio are both 1, and are thus equally effective at eliminating uncorrelated attribute-value combinations.

As seen in the definitions above, when $P(out|\neg c) = 0$ risk ratio is always infinite, so the rate ratio is especially useful when alternative multiple explanations have infinite risk ratio. The same applies to rate ratio when $P(c|\neg out) = 0$.

However, in many use cases, both the risk ratio and the rate ratio can be used to compare multiple explanations and their corresponding outlyingness. Empirically, we have found that the two ratios exhibit similar behavior on many real-world datasets. Figure 15 shows a scatter plot of the risk and rate ratio values for explanations on four of our benchmark datasets: Campaign, Accidents, Telecom, and CMT.

Across all four datasets, the risk and rate ratio distributions behave very similarly, except at the tail for certain pathological cases. For example, in the Telecom dataset, one explanation (the single point on the far right) occurs in the outliers more than 99.99% of the time; thus $b_o/(b_o + b_i)$, the denominator of the risk ratio, is very small. Naturally, the risk ratio here is significantly higher here than the rate ratio (193,838 vs. 3.67). In the CMT dataset, the top right point has a much higher risk ratio than rate ratio due to similar reasons. In practice, we have found that users prefer a wider range of computed ratios to more easily differentiate between explanations; this can potentially inform a user's decision to choose one explanation metric over another.

Overall, alternative explanation metrics, such as the rate ratio, offer MacroBase users an additional technique to better analyze the quality of the underlying explanations for the outliers in their datasets. As these metrics rely on the same statistics, they can be used interchangeably and are key in augmenting MDP.

8.4 Acceleration via Sampling

As data scientists, engineers, and operations experts record increasingly fine-grained detail about the systems they operate and monitor, the dimensionality of input data increases. The primary

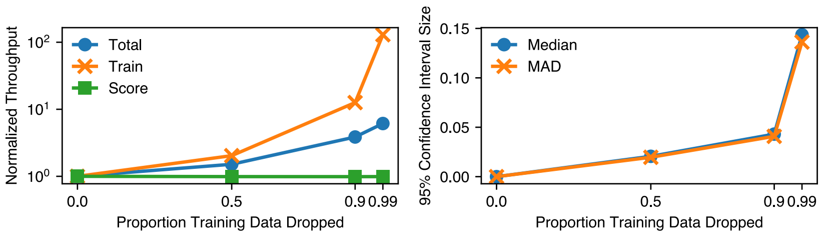


Fig. 16. (Left) Runtime speedup obtained via sampling compared to naive single-metric execution over the CMT dataset (query *MC*). (Right) Size of 95% confidence intervals of the median and MAD estimates, normalized by the true MAD.

challenges that result from this high-dimensional metric data are twofold. First, multivariate results returned via MDP's MCD operator can be difficult to interpret when tens or hundreds of metrics are all highly correlated and contribute to high Mahalanobis distance; for example, all of the servers in a datacenter may experience high CPU and memory utilization during periods of high traffic. In these scenarios, users often struggle to extract meaningful insights from their results even when the contribution of each metric to the Mahalanobis distance is isolated. Second, many metrics are uncorrelated with interesting behavior, and can even simply be noise; for example, in manufacturing and test, it is common to record thousands of readings per device under test, and, in extreme cases system operators may not even know what individual readings correspond to, resulting in wasted computation. As a result, typical MacroBase users routinely execute a suite of single-metric queries to better understand and prune their feature space before running their final MacroBase queries on a limited set of candidate metrics.

The naive approach to automate the process of exploratory feature selection over high-dimensional multivariate data is to run single-metric MacroBase queries over the entire dataset. However, computing over the entire metric space is potentially wasteful, especially as this is an exploratory step before complete analysis. Instead, we developed an alternative means of quickly performing single-metric MacroBase queries via aggressive sampling in conjunction with bootstrapped error bounds, enabling order of magnitude classification-time speedups over the naive approach. Rather than computing the complete single-metric query over each input metric, we modified MDP to optionally sample its input to approximate the true median and MAD, and provide error bounds on these estimates. Once these exploratory, single-metric queries are complete, users then select metrics of interest to run a full MacroBase via MCD. Thus, optimizing this quick exploration as a preprocessing step enables better end-to-end pipeline results for users.

Figure 16 demonstrates the classification-time throughput improvements and average error (size of confidence intervals) from this modified pipeline over the CMT dataset (specifically, query *MC*). Sampling improves train-time throughput (i.e., cost of median and MAD computation) proportional to the amount of data retained, but as the Mahalanobis distance must still be computed for all points, there is no scoring time improvement. We obtain these throughput improvements with relatively small degradation in accuracy, which grows as the number of data points sampled decreases. After a certain point, as throughput improvements are obtained solely from reducing the cost of training, we begin to see diminishing returns with respect to throughput once scoring becomes the bottleneck. We empirically find that sampling 10% of the input data provides an acceptable tradeoff between throughput and accuracy for many datasets. For large datasets or datasets exhibiting low variance, using 1% or less of the input can suffice.

We compute the confidence intervals provided in Figure 16 via bootstrap sampling, which can be expensive. To provide a cheaper proxy, the statistics literature provides many binomial confidence intervals that can be used to provide computationally cheaper confidence interval estimates for the median. The *Agresti-Coull* interval is one of the best, most widely used binomial confidence intervals, and can be applied to obtain a 95%-CI for parameter p (denoted as a range from p_- to p_+), with a sample size of n and X successes as follows [9]:

$$95\%\text{-CI} = \{p_+, p_-\} = \tilde{p} \pm 1.96\sqrt{\frac{1}{\tilde{n}}\tilde{p}(1-\tilde{p})},$$

$$\tilde{n} = n + 1.96^2,$$

$$\tilde{p} = \frac{1}{\tilde{n}}\left(X + \frac{1.96^2}{2}\right).$$

As applied to the median, we set $X = \frac{n}{2}$, resulting in $\tilde{p} = \frac{1}{2}$. To obtain the upper and lower bounds of the confidence interval of the median, we then sort the sampled data points, and return the values at indices corresponding to np_- and np_+ in the sorted list.

The above strategy provides a useful and straightforward modification to MDP. Our ongoing research seeks to further accelerate the feature selection process using more advanced methods that are in turn more computationally and conceptually heavyweight compared to MDP.

9 USAGE AND REFLECTIONS

We sought to design MacroBase as an analytics engine capable of filling the gap between today's low-level dataflow engines and the increasing demands placed on end-user analysts tasked with understanding application and deployment telemetry. In doing so, we made an explicit assumption that dataflow and data storage engines would be commoditized and readily available, and focused our attention on new user interfaces and analytics operators. Today, slightly over 2 years following the inception of the MacroBase project, we have observed several usage patterns that validate our initial hypotheses and inform further research directions.

First, a surprising range of verticals exhibit a demand for fast data analytics—oftentimes, even a few thousand records can be difficult to inspect given sufficiently high dimensionality, and few analysts are willing to manually inspect higher-order feature combinations. While we initially targeted "Internet of Things" scenarios, we believe the core MDP operators are more widely applicable. Via open source and outreach, we have applied MacroBase to domains including mobile devices, cloud/SaaS applications, vehicle monitoring, fabrication, and manufacturing. However, across verticals (and sometimes within verticals), there is a broad range of expertise, ranging from users requiring assistance setting up JDBC connections, to power users who author their own pipelines. Thus, from a systems design perspective, there is considerable work to be done to understand the appropriate means of consuming this technology, whether as a more intelligent stream processor, a Tableau-like exploration tool, or perhaps via new interaction models.

Second, we have found that dataflow and storage capabilities are largely commodity—following the "Big Data" era, many enterprises have made heavy investments into processing engines (e.g., Spark) and ingestion pipelines (e.g., Kafka and HDFS) and are subsequently hungry for new analytics capabilities to utilize this infrastructure. For example, in one production use case at a large Internet service, engineers simply integrated our Java-based stream processing operators as custom user-defined functions in their in-house Java-based stream processing runtime. In fact, at the engineers' request, we published the operators—without the core MacroBase dataflow engine—as a standalone Maven package for deployment.

Nevertheless, modularity and configurability have continued to be a key concern. Beyond the Java-based configuration described in this work, our ongoing work includes a declarative algebra and query planner for MacroBase pipelines, allowing SQL-like pipeline specification and data exploration. Over time, we intend to deprecate the Java-based pipeline construction in favor of this higher-level interface. Moreover, for specialized domains including as time-series and video data, we have found that custom operators that extend MDP (e.g., for visualization [91] and processing [72]) can dramatically boost operator performance and/or efficacy.

Finally, efficient processing remains critical to fast data analytics. We have found that faster processing allows more interactive analytics, and, as analysts and engineers continue to utilize the prototype, a common feature request is to process both more data and higher-dimensional data, faster. As we demonstrated in this article, building an integrated engine allows new optimizations unavailable to operators in isolation, and we believe we have only begun to scratch the surface in terms of optimizations available in practical applications of this technology. More recent results demonstrate it is possible to adapt classic techniques including predicate pushdown, online aggregation, and sketching to the fast data setting, with non-trivial performance implications for common fast data analytics tasks spanning density estimation [51], dimensionality reduction [104], and feature selection [105].

10 RELATED WORK

In this section, we discuss related techniques and systems and compare with our own prior and contemporaneous publications.

Streaming and Specialized Analytics. MacroBase is a data analysis system specialized for prioritizing attention in fast data streams. In its architecture, MacroBase builds upon a long history of systems for streaming data and specialized, advanced analytics tasks. A range of systems from both academia [4, 30] and industry (e.g., Storm, StreamBase, IBM Oracle Streams) provide infrastructure for executing streaming queries. Our MDP pipeline resembles standard data analytics pipelines, including stages for ingest, preprocessing, and classification. While MacroBase adopts dataflow as its execution substrate, its goal is to provide a set of high-level analytic monitoring operators; in MacroBase, dataflow is a means to an end rather than an end in itself. In designing a specialized engine, we were inspired by several past projects, including Gigascope (specialized for network monitoring) [39], WaveScope (specialized for signal processing) [53], MCDB (specialized for Monte Carlo-based operators) [71], and Bismarck (providing extensible aggregation for gradient-based optimization) [47].

In addition, a range of commercially available analytics packages provide advanced analytics functionality. For example, MOA [20] supports a collection of supervised methods for learning from data streams, while SAMOA [40] further extends the functionalities to distributed environments. MacroBase continues this tradition by providing a specialized set of operators for classification and explanation of fast data. Analogous to the decision tree and Bayesian classifiers in MOA, MacroBase provides stream classifications operators based on distribution modeling; however, MacroBase significantly improves the interpretability of the classification results by introducing streaming explanation operators that succinctly summarize the key outlying subpopulations. While MacroBase's streaming explanation shares similarities with sample-based frequent itemset mining techniques [90] implemented in SAMOA, MacroBase exhibits order-of-magnitude performance improvements through the novel optimizations enabled by the co-design of classification and explanation in its analytic pipeline.

Classification. Classification and outlier detection have an extensive history; the literature contains thousands of techniques from communities including statistics, machine learning, data mining, and information theory [7, 25, 28, 63]. Outlier detection techniques have seen major success in several domains including network intrusion detection [45, 83], fraud detection (leveraging a variety of classifiers and techniques) [21, 87], and industrial automation and predictive maintenance [10, 78]. A considerable subset of these techniques operates over data streams [6, 27, 88, 103].

At stream volumes in the hundreds of thousands or more events per second, statistical outlier detection techniques produce a large stream of outlying data points by nature. As a result, while outlier detection forms a core component of a fast data analytics engine, it must be coupled with streaming explanation. In the design of MacroBase, we treat the array of classification techniques as inspiration for a modular architecture. In MacroBase's default pipeline, we leverage detectors based on robust statistics [65, 79] adapted to the streaming context. However, we also demonstrate compatibility with detectors from Elki [100], Weka [56], RapidMiner [64], and OpenGamma [1]. Since the MacroBase design can explain issues independent of a specific outlier classification technique, our simple methods are able to achieve high performance while retaining result quality.

MAD and MCD are efficient and robust methods within the family of (generative) distribution-modeling–based outlier classification techniques. Their use of an underlying distribution model improves interpretability when users want to understand the relation of outliers to other points in the data stream. Other distribution-modeling methods include Kernel density estimation [85] and Gaussian mixture models [61], which tend to be more computationally expensive. Other methods for improving robustness include Huber's method and RANSAC [48, 93]. We found the simpler MAD/MCD models sufficient in our evaluations and incorporating more advanced robust statistics are a rich direction for future research.

Unlike distribution-based outlier detection methods, some alternative techniques focus only on classifying points: this includes Local Outlier Factor, DBScan, one-class SVMs, and a host of data mining techniques [22, 25, 46, 99]. Algorithms such as Hierarchical Temporal Memory [11] do not classify outlying data points in isolation, but identify contextual outliers based on the sequence of points in a data stream. Furthermore, additional techniques focus on identifying collective outliers based on data segments [108] or contextual, temporal outliers [55]. These are powerful tools that can be incorporated within the MacroBase pipeline based on application-specific needs.

One of the key challenges of applying these detectors in the streaming context is the ability to quickly adapt to changes in the input distribution. In fact, developing methods that are sensitive to the time-varying nature of the input, or concept drift, while remaining robust to noise has been an area of active research [50, 98, 116]. MacroBase utilizes ADR, a novel adaptation of reservoir sampling over streams, to periodically retrain its detectors as well as to adjust the outlier detection thresholds. Our evaluation showed that the ADR can quickly adapt to systemic shifts in input and is robust to short noise spikes under variable data rates. However, MacroBase does not store or reuse previous experience when old scenarios reoccur, for example, due to seasonality [110]. For future work, we hope to learn and track distinct states of the input, both for improving detection accuracy and for better explaining deviations in behavior.

Data Explanation. Data explanation techniques assist in summarizing differences between datasets. The literature contains several recent explanation techniques leveraging decision-tree [32] and Apriori-like [57, 112] pruning, grid search [94, 114], data cubing [95], Bayesian statistics [109], visualization [26, 84], causal reasoning [113], and several others [43, 62, 70, 80]. While we are inspired by these results, none of these techniques executes over streaming data or at the scale we seek. Several exhibit runtime exponential in the number of attributes (which can number in the hundreds of thousands to millions in the fast data we examine) [95, 109] and, when reported, runtimes in the batch setting often vary from hundreds to approximately 10K points per second [94, 109, 112] (we also directly compare throughput with several techniques [32, 95, 109, 112] in Section 7.5).

To address the demands of streaming operation and to scale to millions of events per second, MacroBase's explanation techniques draw on sketching and streaming data structures (specifically [31, 33, 36, 38, 41, 81, 97, 106]), adapted to the fast data setting. We view existing explanation techniques as a useful second step in analysis following the explanations generated by MacroBase, and we see promise in adapting these existing techniques to streaming execution at high volume. Given our goal of providing a generic architecture for analytic monitoring, future improvements in streaming explanation should be complementary to our results here.

Prior Publications. As mentioned in Section 1, this invited article includes material from and significantly extends [16]. This earlier publication introduced the core MacroBase architecture and MDP pipeline as of approximately ten months of work on the system prototype. In addition, [17] provides a gentle overview of the MacroBase system, focusing on our initial design philosophy, and reports on different user interfaces provided by the system. This article extends our prior work in several directions, leveraging the benefit of over a year of additional experience with the MacroBase engine. We present and evaluate four extensions of the MDP pipeline as informed by early use cases: operation over aggregate and pre-cubed input data (Section 8.1), sliding-window explanation (Section 8.2), support for alternative explanation metrics (Section 8.3), and lightweight sample-based feature selection (Section 8.4). Each of these features is inspired by a real-world use case, and each necessitates non-trivial adaptations of the MDP pipeline. In addition, we provide a discussion of ongoing use cases and engine usage patterns, reflecting on our initial architectural design decisions and impact on ongoing research (Section 9).

As a systems research project, MacroBase is the vehicle for a range of ongoing projects. Following the above conference publications, we have recently authored several related articles addressing different extensions to MacroBase; for completeness, we briefly compare each of these articles here. MDP's explanation operator presented in this work (Section 5) processes categorical data; in contrast, our recent work on ASAP [91] provides an alternative explanation operator for time-series and was inspired by the electricity case study in Section 7.4. In addition, the video surveillance case study in Section 7.4 uses classical computer vision feature transformation (i.e., optical flow) before MDP classification; in contrast, our recent work on NoScope [72] demonstrates how to efficiently utilize neural networks for more sophisticated video featurization. MDP leverages parametric density estimators (MAD and MCD) in classification (Section 4); in contrast, our recent work on tKDC [51] illustrates how to provide efficient non-parametric kernel density estimates in classification. MDP uses sample-based feature selection routines for exploratory feature selection over high-dimensional data (Section 8.4); our ongoing work on online aggregation [104] provides an alternative for high-dimensional time-series data. Finally, while we use Heavy-Hitters sketches for efficient explanation in MDP (Section 5), our recent work [105] illustrates how to utilize Count-Min data structures to instead perform this computation.

Each of the above publications represents a new feature transformation, classification, or explanation operator that can interoperate with MDP's default operators and may, over time, replace MDP's operators. Thus, these recent articles are complementary to the goals of our work here, which introduces the MacroBase architecture and MDP pipeline operators.

11 CONCLUSIONS AND FUTURE WORK

We have presented MacroBase, a new analytics engine to prioritize attention in fast data streams. MacroBase provides a flexible architecture that delivers interpretable summaries of important behavior in fast data streams by combining streaming classification and data explanation techniques. MacroBase's default analytics operators, which include new sampling and sketching procedures, take advantage of this combination of detection and explanation and are optimized for high-volume, time-sensitive, and heterogeneous data streams, resulting in improved performance and result quality. This emphasis on flexibility, accuracy, and speed has proven useful in several production deployments, where MacroBase has already identified previously unknown behaviors.

MacroBase is available as open source and is under active development. The system serves as the vehicle for a number of ongoing research efforts, while ongoing production use cases continue to stimulate the development of new functionality that can leverage the flexibility provided by MacroBase's architecture.

REFERENCES

[1] 2016. OpenGamma. Retrieved July 2016 from http://www.opengamma.com/.
[2] 2016. Summary of the Amazon DynamoDB Service Disruption and Related Impacts in the US-East Region. Retrieved February 2016 from https://aws.amazon.com/message/5467D2/.
[3] 2017. CAVIAR Test Case Scenarios. Retrieved November 2017 from http://homepages.inf.ed.ac.uk/rbf/CAVIAR/.
[4] Daniel J. Abadi et al. 2005. The design of the Borealis stream processing engine. In *CIDR*.
[5] Charu C. Aggarwal. 2006. On biased reservoir sampling in the presence of stream evolution. In *VLDB*.
[6] Charu C. Aggarwal. 2007. *Data Streams: Models and Algorithms*. Vol. 31.
[7] Charu C. Aggarwal. 2013. *Outlier Analysis*.
[8] Shikha Agrawal and Jitendra Agrawal. 2015. Survey on anomaly detection using data mining techniques. *Procedia Computer Science* 60 (2015), 708–713.
[9] Alan Agresti and Brent A. Coull. 1998. Approximate is better than "exact" for interval estimation of binomial proportions. *The American Statistician* 52, 2 (1998), 119–126.
[10] Rosmaini Ahmad and Shahrul Kamaruddin. 2012. An overview of time-based and condition-based maintenance in industrial application. *Computers & Industrial Engineering* 63, 1 (2012), 135–149.
[11] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. 2017. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing* 262 (2017), 134–147.
[12] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. 2016. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications* 60 (2016), 19–31.
[13] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: A survey. *Data Mining and Knowledge Discovery* 29, 3 (2015), 626–688.
[14] Arvind Arasu, Shivnath Babu, and Jennifer Widom. 2006. The CQL continuous query language: Semantic foundations and query execution. *The VLDB Journal* 15, 2 (2006), 121–142.
[15] Anthony Asta. 2016. Observability at Twitter: Technical overview, part I. Retrieved from https://blog.twitter.com/2016/observability-at-twitter-technical-overview-part-i.
[16] Peter Bailis, Edward Gan, Samuel Madden, Deepak Narayanan, Kexin Rong, and Sahaana Suri. 2017. MacroBase: Prioritizing attention in fast data. In *SIGMOD*.
[17] Peter Bailis, Edward Gan, Kexin Rong, and Sahaana Suri. 2017. Prioritizing attention in fast data: Principles and promise. In *CIDR*.
[18] Christian Beckel et al. 2014. The ECO data set and the performance of non-intrusive load monitoring algorithms. In *BuildSys*. ACM.
[19] Yoav Benjamini and Daniel Yekutieli. 2001. The control of the false discovery rate in multiple testing under dependency. *Annals of Statistics* (2001), 1165–1188.
[20] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. 2010. MOA: Massive online analysis. *Journal of Machine Learning Research* 11 (2010), 1601–1604.
[21] Richard J. Bolton and David J. Hand. 2002. Statistical fraud detection: A review. *Statistical Science* (2002), 235–249.
[22] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: Identifying density-based local outliers. In *SIGMOD*. 93–104.
[23] Chiranjeeb Buragohain and Subhash Suri. 2009. Quantiles on streams. In *Encyclopedia of Database Systems*. 2235–2240.
[24] R. W. Butler, P. L. Davies, and M. Jhun. 1993. Asymptotics for the minimum covariance determinant estimator. *The Annals of Statistics* (1993), 1385–1400.
[25] Guilherme O. Campos, Arthur Zimek, Jörg Sander, Ricardo J. Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E. Houle. 2016. On the evaluation of unsupervised outlier detection: Measures, datasets, and an empirical study. *Data Miing and Knowedge Discovery* 30, 4 (July 2016), 891–927.
[26] Lei Cao, Qingyang Wang, and Elke A. Rundensteiner. 2014. Interactive outlier exploration in big data streams. *VLDB* 7, 13 (2014), 1621–1624.

[27] Lei Cao, Di Yang, Qingyang Wang, Yanwei Yu, Jiayuan Wang, and Elke A. Rundensteiner. 2014. Scalable distance-based outlier detection over high-volume data streams. In *ICDE*.
[28] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)* 41, 3 (2009), 15.
[29] Badrish Chandramouli, Jonathan Goldstein, et al. 2014. Trill: A high-performance incremental query processor for diverse analytics. In *VLDB*.
[30] Sirish Chandrasekaran et al. 2003. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*.
[31] MT Chao. 1982. A general purpose unequal probability sampling plan. *Biometrika* 69, 3 (1982), 653–656.
[32] Mike Chen, Alice X. Zheng, Jim Lloyd, Michael I. Jordan, and Eric Brewer. 2004. Failure diagnosis using decision trees. In *ICAC*.
[33] James Cheng et al. 2008. A survey on algorithms for mining frequent itemsets over data streams. *Knowledge and Information Systems* 16, 1 (2008), 1–27.
[34] Kai-Wen Cheng, Yie-Tarng Chen, and Wen-Hsien Fang. 2015. Video anomaly detection and localization using hierarchical feature representation and Gaussian process regression. In *CVPR*. 2909–2917.
[35] David Clayton, Michael Hills, and A. Pickles. 1993. *Statistical Models in Epidemiology*. Vol. 161. Oxford University Press, Oxford.
[36] Graham Cormode, Minos Garofalakis, Peter J. Haas, and Chris Jermaine. 2012. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases* 4, 1–3 (2012), 1–294.
[37] Graham Cormode and Marios Hadjieleftheriou. 2010. Methods for finding frequent items in data streams. *The VLDB Journal* 19, 1 (2010), 3–20.
[38] Graham Cormode, Flip Korn, and Srikanta Tirthapura. 2008. Exponentially decayed aggregates on data streams. In *ICDE*.
[39] Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk. 2003. Gigascope: A stream database for network applications. In *SIGMOD*.
[40] Gianmarco De Francisci Morales and Albert Bifet. 2015. SAMOA: Scalable advanced massive online analysis. *Journal of Machine Learning Resarche* 16, 1 (2015), 149–153.
[41] Pavlos S. Efraimidis. 2015. Weighted random sampling over data streams. In *Algorithms, Probability, Networks, and Games*. Springer, 183–195.
[42] Alexei A. Efros, Alexander C. Berg, Greg Mori, and Jitendra Malik. 2003. Recognizing action at a distance. In *ICCV*.
[43] Kareem El Gebaly, Parag Agrawal, Lukasz Golab, Flip Korn, and Divesh Srivastava. 2014. Interpretable and informative explanations of outcomes. In *VLDB*.
[44] Sarah M. Erfani, Sutharshan Rajasegarar, Shanika Karunasekera, and Christopher Leckie. 2016. High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning. *Pattern Recognition* 58 (2016), 121–134.
[45] Terry Escamilla. 1998. *Intrusion Detection: Network Security Beyond the Firewall*. John Wiley & Sons, Inc.
[46] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*. 226–231.
[47] Xixuan Feng, Arun Kumar, Benjamin Recht, and Christopher Ré. 2012. Towards a unified architecture for in-RDBMS analytics. In *SIGMOD*.
[48] Martin A. Fischler and Robert C. Bolles. 1987. Readings in computer vision: Issues, problems, principles, and paradigms. Chapter: Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography, 726–740.
[49] Philippe Fournier-Viger. 2015. SPMF: An Open-Source Data Mining Library – Performance. Retrieved from http://www.philippe-fournier-viger.com/spmf/.
[50] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM Computing Surveys* 46, 4 (March 2014), Article 44, 37 pages.
[51] Edward Gan and Peter Bailis. 2017. Scalable kernel density classification via threshold-based pruning. In *SIGMOD*.
[52] Paul H. Garthwaite and Inge Koch. 2016. Evaluating the contributions of individual variables to a quadratic form. *Australian & New Zealand Journal of Statistics* 58, 1 (2016), 99–119.
[53] Lewis Girod et al. 2006. Wavescope: A signal-oriented data stream management system. In *ICDE*.
[54] Markus Goldstein and Seiichi Uchida. 2016. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLoS One* 11, 4 (04 2016), 1–31.
[55] Manish Gupta, Jing Gao, Charu Aggarwal, and Jiawei Han. 2014. Outlier detection for temporal data. *Synthesis Lectures on Data Mining and Knowledge Discovery* 5, 1 (2014), 1–129.
[56] Mark Hall et al. 2009. The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter* 11, 1 (2009), 10–18.
[57] Jiawei Han et al. 2007. Frequent pattern mining: Current status and future directions. *Data Mining and Knowledge Discovery* 15, 1 (2007), 55–86.

[58] Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining frequent patterns without candidate generation. In *SIGMOD*.

[59] Johanna Hardin and David M. Rocke. 2004. Outlier detection in the multiple cluster setting using the minimum covariance determinant estimator. *Computational Statistics & Data Analysis* 44, 4 (2004), 625–638.

[60] Johanna Hardin and David M. Rocke. 2005. The distribution of robust distances. *Journal of Computational and Graphical Statistics* 14, 4 (2005), 928–946.

[61] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning*. Springer-Verlag, New York.

[62] Joseph M. Hellerstein. 2008. Quantitative data cleaning for large databases. *United Nations Economic Commission for Europe (UNECE)*.

[63] Victoria J. Hodge and Jim Austin. 2004. A survey of outlier detection methodologies. *Artificial Intelligence Review* 22, 2 (2004), 85–126.

[64] Markus Hofmann and Ralf Klinkenberg. 2013. *RapidMiner: Data Mining Use Cases and Business Analytics Applications*. CRC Press.

[65] Peter J. Huber. 2011. *Robust Statistics*. Springer.

[66] Mia Hubert and Michiel Debruyne. 2010. Minimum covariance determinant. *Wiley Interdisciplinary Reviews: Computational Statistics* 2, 1 (2010), 36–43.

[67] William H. Hunter. 2015. US Chemical Safety Board: Analysis of Horsehead Corporation Monaca Refinery Fatal Explosion and Fire. Retrieved from http://www.csb.gov/horsehead-holding-company-fatal-explosion-and-fire/.

[68] J.-H. Hwang, Magdalena Balazinska, et al. 2005. High-availability algorithms for distributed stream processing. In *ICDE*.

[69] IDC. 2014. The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things. Retrieved from http://www.emc.com/leadership/digital-universe/.

[70] Ihab F. Ilyas and Xu Chu. 2015. Trends in cleaning relational data: Consistency and deduplication. *Foundatations and Trends in Databases* 5, 4 (2015), 281–393.

[71] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher Jermaine, and Peter J. Haas. 2008. MCDB: A Monte Carlo approach to managing uncertain data. In *SIGMOD*.

[72] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing neural network queries over video at scale. In *VLDB*.

[73] Yannis Klonatos, Christoph Koch, Tiark Rompf, and Hassan Chafi. 2014. Building efficient query engines in a high-level language. In *VLDB*.

[74] Haiquan Li, Jinyan Li, Limsoon Wong, Mengling Feng, and Yap-Peng Tan. 2005. Relative risk and odds ratio: A data mining perspective. In *PODC*.

[75] Jin Li, David Maier, Kristin Tufte, Vassilis Papadimos, and Peter A. Tucker. 2005. No pane, no gain: Efficient evaluation of sliding-window aggregates over data streams. *SIGMOD Record* 34, 1 (March 2005), 39–44.

[76] Jessica Lin et al. 2005. Visualizing and discovering non-trivial patterns in large time series databases. *Information Visualization* 4, 2 (2005), 61–82.

[77] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. 2015. Long short term memory networks for anomaly detection in time series. In *European Symposium on Artificial Neural Networks*. 89.

[78] Riccardo Manzini, Alberto Regattieri, Hoang Pham, and Emilio Ferrari. 2009. *Maintenance for Industrial Systems*. Springer Science & Business Media.

[79] RARD Maronna, Douglas Martin, and Victor Yohai. 2006. *Robust Statistics*. John Wiley & Sons, Chichester. ISBN.

[80] Alexandra Meliou, Sudeepa Roy, and Dan Suciu. 2014. Causality and explanations in databases. In *VLDB*.

[81] Ahmed Metwally et al. 2005. Efficient computation of frequent and top-k elements in data streams. In *ICDT*.

[82] Julie A. Morris and Martin J. Gardner. 1988. Statistics in medicine: Calculating confidence intervals for relative risks (odds ratios) and standardised ratios and rates. *British Medical Journal (Clinical Research ed.)* 296, 6632 (1988), 1313.

[83] Biswanath Mukherjee, L. Todd Heberlein, and Karl N. Levitt. 1994. Network intrusion detection. *IEEE Network* 8, 3 (1994), 26–41.

[84] Vinod Nair et al. 2015. Learning a hierarchical monitoring system for detecting and diagnosing service issues. In *KDD*.

[85] Emanuel Parzen. 1962. On estimation of a probability density function and mode. *Annals of Mathematical Statistics* 33, 3 (1962), 1065–1076.

[86] Tuomas Pelkonen et al. 2015. Gorilla: A fast, scalable, in-memory time series database. In *VLDB*.

[87] Clifton Phua, Vincent Lee, Kate Smith, and Ross Gayler. 2010. A comprehensive survey of data mining-based fraud detection research. *arXiv preprint arXiv:1009.6119*

[88] Dragoljub Pokrajac, Aleksandar Lazarevic, and Longin Jan Latecki. 2007. Incremental local outlier detection for data streams. In *CIDM*.

[89] Benjamin Reiser. 2001. Confidence intervals for the Mahalanobis distance. *Communications in Statistics-Simulation and Computation* 30, 1 (2001), 37–45.

[90] Matteo Riondato, Justin A. DeBrabant, Rodrigo Fonseca, and Eli Upfal. 2012. PARMA: A parallel randomized algorithm for approximate association rules mining in MapReduce. In *CIKM*. 85–94.

[91] Kexin Rong and Peter Bailis. 2017. ASAP: Prioritizing attention via time series smoothing. In *VLDB*.

[92] Peter J. Rousseeuw and Katrien Van Driessen. 1999. A fast algorithm for the minimum covariance determinant estimator. *Technometrics* 41, 3 (1999), 212–223.

[93] Peter J. Rousseeuw and Mia Hubert. 2011. Robust statistics for outlier detection. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1, 1 (2011), 73–79.

[94] Sudip Roy, Arnd Christian König, Igor Dvorkin, and Manish Kumar. 2015. Perfaugur: Robust diagnostics for performance anomalies in cloud services. In *ICDE*.

[95] Sudeepa Roy and Dan Suciu. 2014. A formal approach to finding explanations for database queries. In *SIGMOD*.

[96] G. Rupert, Jr. et al. 2012. *Simultaneous Statistical Inference*. Springer Science & Business Media.

[97] Florin I. Rusu. 2009. *Sketches for Aggregate Estimations Over Data Streams*. Ph.D. dissertation. University of Florida.

[98] Jeffrey C. Schlimmer and Richard H. Granger, Jr. 1986. Incremental learning from noisy data. *Machine Learning* 1, 3 (March 1986), 317–354.

[99] Bernhard Schölkopf, John C. Platt, John C. Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. 2001. Estimating the support of a high-dimensional distribution. *Neural Computation* 13, 7 (July 2001).

[100] Erich Schubert, Alexander Koos, Tobias Emrich, Andreas Züfle, Klaus Arthur Schmid, and Arthur Zimek. 2015. A framework for clustering uncertain data. In *VLDB*.

[101] W. Shi and B. M. Golam Kibria. 2007. On some confidence intervals for estimating the mean of a skewed population. *International Journal of Mathematical Education in Science and Technology* 38, 3 (2007), 412–421.

[102] Herbert A. Simon. 1971. Designing organizations for an information rich world. In *Computers, Communications, and the Public Interest*. 37–72. http://opacplus.bsb-muenchen.de/search?isbn=0-8018-1135-X.

[103] Sharmila Subramaniam et al. 2006. Online outlier detection in sensor data using non-parametric models. In *VLDB*.

[104] Sahaana Suri and Peter Bailis. 2017. DROP: Dimensionality reduction optimization for time series. *arXiv:1708.00183*. https://arxiv.org/pdf/1708.00183.pdf.

[105] Kai Sheng Tai, Vatsal Sharan, Peter Bailis, and Gregory Valiant. 2018. Sketching linear classifiers over data streams. In *SIGMOD*. 757–772.

[106] Syed Khairuzzaman Tanbeer et al. 2009. Sliding window-based frequent pattern mining over data streams. *Information Sciences* 179, 22 (2009), 3843–3865.

[107] Jeffrey S. Vitter. 1985. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)* 11, 1 (1985), 37–57.

[108] Xiang Wang and Ian Davidson. 2009. Discovering contexts and contextual outliers using random walks in graphs. In *ICDM*. 1034–1039.

[109] Xiaolan Wang, Xin Luna Dong, and Alexandra Meliou. 2015. Data x-ray: A diagnostic tool for data errors. In *SIGMOD*.

[110] Gerhard Widmer and Miroslav Kubat. 1996. Learning in the presence of concept drift and hidden contexts. *Machine Learning* 23, 1 (April 1996), 69–101.

[111] Alex Woodie. 2015. Kafka tops 1 trillion messages per day at LinkedIn. *Datanami* (September 2015). http://www.datanami.com/2015/09/02/kafka-tops-1-trillion-messages-per-day-at-linkedin/.

[112] Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining away outliers in aggregate queries. In *VLDB*.

[113] Dong Young Yoon, Ning Niu, and Barzan Mozafari. 2016. DBSherlock: A performance diagnostic tool for transactional databases. In *SIGMOD*.

[114] Ziming Zheng, Yawei Li, and Zhiling Lan. 2007. Anomaly localization in large-scale clusters. In *ICCC*.

[115] Chong Zhou and Randy C. Paffenroth. 2017. Anomaly detection with robust deep autoencoders. In *SIGKDD*. 665–674.

[116] Indre Zliobaite. 2010. Learning under concept drift: An overview. *CoRR* abs/1010.4784 (2010). http://arxiv.org/abs/1010.4784

Received November 2017; revised July 2018; accepted September 2018