Truong, Linh; Nhu Trang, Nguyen Ngoc

Analytics Feature Space: a Novel Framework for Interoperable Edge Machine Learning Detection

Submitted: 29/03/2024

Analytics Feature Space: a Novel Framework for Interoperable Edge Machine Learning Detection

Hong-Linh Truong
Aalto University, Finland
linh.truong@aalto.fi

Nguyen Ngoc Nhu Trang
Daienso Lab, Vietnam
nhutrang.nguyen@daienso.com

***Abstract*—Departure from the analytics for single assets, in many application domains, complex scenarios require us to determine and discover facts and insights for a collective of assets, spaces, and environments. Due to the advance of machine learning (ML), such complex scenarios are increasingly relied on different edge ML detection, which are diverse for different purposes, like object detection, anomaly detection, surface defect detection or activity recognition/classification, in a unified management view of a business. Widely integrated and provisioned at the edge, the diversity of the underlying ML models, concrete deployments, and changes in operations lead to different types of detection results. However, extensive metadata required for interpreting the results is not well supported, in addition to the difficulty when integrating and analyzing detection results from multiple pipelines and algorithms. In this paper, we present Analytics Feature Space (AFS) as a novel framework to support high-level analytics and integration for multiple types of ML detection at the edge in a unified way that is applied to collectives of assets, spaces and environments. AFS abstracts and supports key metadata related to edge ML detection, changes in detection deployments and metadata for the above-mentioned collective detection and analytics. We introduce techniques to manage relationships between analytics subjects, ML detection models and results. Different ways for integrating AFS and detection pipelines are presented. We demonstrate our experiments with realistic scenarios for operation management.**

I. Introduction

Currently, various ML pipelines have been developed and deployed at the edge to carry out different types of near real-time detection. Although object detection [1], [2], [3], [4] is widely used in many applications, in real-world complex systems like Intelligent Operation Centers for industrial zones or Digital Twins, we have many other types of detection, like anomaly detection [5], [6], [7], event detection, quality control detection of specific categories, hazardous material and surface detection. In parallel, in the view of analytics for different purposes, we move away from single detection for single assets to the analytics of complex subjects encapsulating highly connected assets (e.g., pumps and pipes), related spaces (e.g., wastewater lakes), and environments (e.g., wastewater and air) in a managed business. We call entities abstracted in such a complex subject that must be managed together a Collective of Assets, Spaces, and Environments (CASE). In the interest of complex operations and management, to manage CASE, the analytics can leverage many types of above-mentioned detection[1]. Thus, there is a need to integrate different types of detection for a holistic analytics of CASE.

To date, the results from the detection at the edge are normally delivered to the downstream (cloud-based) applications via messaging systems to support near real-time stream analytics and decision-making and to store the results into data lakes for future analytics. When examining the detection needed for the analytics of CASE, two key observations are drawn. First, the detection pipelines are diverse in terms of underlying ML models (e.g., object detection models and time series anomaly data detection models) and outputs (e.g., multiple object detection vs specific types of anomaly detection). Second, due to highly decoupling and ML model changes in software and system operations at the edge, these pipelines can be reconfigured and changed w.r.t. runtime execution configurations, ML model configurations, and data sources without controls/awareness of the downstream applications/systems. Therefore, for complex applications in the downstream, which aggregate the results from various pipelines, there are many types of data that can be missed or inadequate to support the analytics and integration of the outputs from these pipelines.

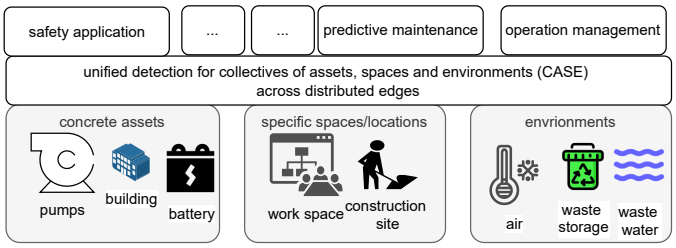

Fig. 1. Unified detection management for collectives of assets, spaces, and environments at the edge. Often the operator needs applications to analyze multiple related entities in a CASE.

Most state-of-the-art systems support tightly integrated single, specific detection pipelines for specific purposes, such as traffic management, face detection, and surveillance, within a view of Internet of Things (IoT) platforms. Thus, existing techniques do not consider dynamic changes due to the decoupling of the operation of these edge pipelines in complex systems,

[1]Although in ML, "detection" has a specific interpretation, distinguishable from, e.g., classification or recognition. This paper uses the term "detection" in a broader sense to indicate the fact that, for management purposes, we need to discover and determine insightful states and facts of target entities.

as mentioned above. However, real-world applications need to deal with the management of multiple types of detection applied to CASE, as show in Figure 1. In our focus, downstream applications often require many results from different detection pipelines without having a complete knowledge about changes at the edge. For such applications, it is important to obtained various types of metadata, besides the detection results, as well as changes in order to carry out the analytics. This requires us to carefully devise new types of traceable, metadata and changes that must be captured and delivered together with the detection results to support downstream applications. In this work, we address the above-mentioned issues by introducing the Analytics Feature Space (AFS) as a novel framework to address the interoperability, integration and explainability for multiple types of ML detection at the edge in a unified way for CASE. The paper makes the following contributions:

- making several types of metadata associated with edge detection pipelines for a CASE explicitly: many types of metadata are implicitly hidden inside the implementation; they are not available or hard to obtain for downstream analytics. This paper identifies and introduces specifications for many metadata that must be explicitly modeled.
- capturing important changes for downstream applications: given the autonomous deployment and operation of edge pipelines, cloud-based downstream applications must be aware of the changes in order to make decisions. This paper contributes techniques for capturing such data, powering metadata enrichment for detection results.
- providing a generic framework with functions and middleware for the enrichment of results and metadata: usually such enrichment cannot be done at a single place due to the complex deployment. This paper develops various functions together with messaging middleware and other relevant components to support the enrichment.

The AFS framework supports analytics subjects for CASE in a flexible and reusable way. By utilizing the AFS, many types of necessary metadata can be provided, together with different results of detection models that are transformed and enriched into common schemas and categories, including domain and optional information. This will help the development of downstream analytics and detection result sharing.

The rest of this paper is organized as follows: Section II presents our scenarios and requirements. We present the AFS framework in Section III and provide the implementation detail in Section IV. Section V presents examples and experiments. We discuss the related work in Section VI and conclude the paper in Section VII.

II. Scenarios and Research Questions

A. Scenarios

Let us consider a scenario in which multiple, different upstream detection pipelines at the edge will provide detection services to the downstream applications for CASE. Our scenario is based on a realistic Intelligent Operation Center (IOC) for industrial zones and factories, especially in developing countries. The exemplified IOC type is to manage environmental conditions and safety in different wastewater processing factories located in different places. Several edge cameras and IoT devices are deployed, managed, and operated autonomously with ML-based detection pipelines, decoupled from the IOC services in the cloud. The analytics of CASE for IOC requires:

- edge detection (upstream detection): data from cameras and IoT devices are fed into ML pipelines for various types of detection, such as restricted area violation detection (using cameras) or wastewater surface detection (using cameras) or wastewater monitoring (using sensors for oxygen, pH and pressure measurement). The *relevant, significant* detection results will also be sent to the IOC services in the cloud. However, not all (detailed) results are sent, but follow the configuration, e.g., periodically every hour, or only for important events, measurements, objects, and possible raw images/detailed evidence if some conditions are met. During the operation, these pipelines can be updated, changed, and reconfigured independently from the cloud-based IOC services. Thus, if not updated, the IOC will not know the changes, which can affect the management and decision-making.
- downstream applications for the analytics of CASE: based on the high-level results from various upstream detection, further detection and analytics will be carried out. For example, for safety purposes, an analytics may require upstream object detection and restricted area violation detection, detection of wastewater lakes quality from wastewater surface detection and wastewater monitoring. Such analytics and further detection reflect a composable (real-time) monitoring and warning system that integrates upstream results to improve the efficiency of CASE management using multiple detection.

Although upstream detection may be distributed across different places, all of them are for CASE that must be managed in a unified view to serve complex applications:

1) An safety application needs a unified view of upstream object detection and anomaly detection: Common characteristics of factories are their large area and many surveillance needs, such as safety and emergency situation support. Besides single pipelines for specific places, such as surveillance detection in wastewater processing lakes or activity detection in office areas, unified monitoring for the whole factory, especially for emergency support, is needed. For example, where does the emergent situation occur? How many people are in that dangerous area? Checking all cameras or every single detection result at that time is not an optimal solution. Therefore, integrating detection pipelines to have a unified view and sharing detected data is necessary in an IOC or in some emergency situations.

2) A quality control application needs a unified view of upstream anomaly detection and activity detection: Automatic quality control via images or video is carried out for water lakes by detecting anomalies in areas that are required to be monitored all the time or are difficult for humans to access.

Drones or specialized cameras must be used instead. Moreover, in the case of semi-automatic wastewater processing, where workers take roles in some parts of the processing, activity detection can be deployed. Together with camera system, the quality control also uses sensors to monitor water quality, pressure, temperature, to name just a few. In order to build an overall quality control monitoring application, the IOC needs to integrate and combine different types of measurements and deploy many downstream analytics.

B. Research Questions (RQs)

From the scenarios, two RQs are identified that need to be tackled:

RQ1 – Which types of metadata and how to link them to the detection results and analytics for a CASE? We must address the diversity of ML detection pipelines by modeling and capturing common metadata reflecting different types of detection, ML models and assets in a CASE, such as detection classes, models, and data sources. Furthermore, metadata should help us to integrate and route feature instances resulted from detection pipelines at the edge for downstream applications in a unified view. Many more types of metadata about ML inferences/detection must be provided to support the use of detection results, including explainability. Given the metadata provided by the pipeline, downstream applications must be able to access further metadata about ML algorithms to make decision. Furthermore, metadata enrichment functions must be supported to enhance detection results to adapt to downstream analytics needs.

RQ2 – What types of changes need to be captured and exchanged? Many changes can happen in detection models and their configurations due to model updates or ensemble configurations. At runtime, changes in detection result constraints can be observed as ML pipelines can be highly configurable to specify constraints on detection results to be delivered to the downstream. Furthermore, changes in data sources associated with analytics subject occur when one or more data sources for a pipeline are added or removed. These changes are triggered by two main situations: the context awareness of the business/deployment (e.g., the minimum thresholds for a specific CASE) and the runtime adaptation due to adaptive control strategies (e.g., reducing detection rates). Especially, changes in constraints for detection results that are based on runtime contexts and deployment contexts should be communicated to avoid misunderstanding of the quality of detection results and to support explainability on data drifts in detection.

III. The Analytics Feature Space framework

A. Defining an Analytics Feature Space: unified detection management for CASE at the edge

We first explain our unified view for multi types of detection at the edge in our focus. In existing work, typically there exist multiple assets (e.g., pumps and pipes) and each asset has a set of dedicated monitoring points, e.g., linked to sensors/cameras, for the asset only. Then, analytics/detection is specifically applied to the asset by analyzing measurements and detection results from data provided by these monitoring points. However, due to the dynamic edge deployments, software, changes and the concept of analytics/detection associated with the domain, several relationships are complex and changing. For example, a monitoring point at a specific location can provide data for a concrete asset located in that location, such as an equipment, but can also provide data for monitoring of the specific location. Hence a single monitoring point is for two separated different analytics subjects.

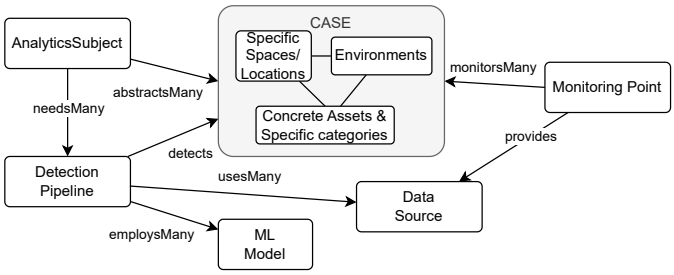

Fig. 2. High-level conceptual relationships of main elements in a unified detection view.

Figure 2 presents a unified view of relevant software and data components for dynamic, multiple types of detection. From the application viewpoint, a *Detection Pipeline* can be deployed for detection of an asset, space or environment. A *Detection Pipeline* will rely on *Data Source* provided by *Monitoring Point*. An *Analytics Subject* is a high-level abstraction that indicates the subject to be analyzed is a CASE. The subject can include a concrete asset (e.g., like a pump or a building), a specific location, entries to gates or wastewater lake, or a specific category like wastewater. Analytics subjects are usually defined in the application specific setting, such as downstream analytics. To analyze a subject, we use different ML models with different data sources and techniques. However, analytics subjects and data sources have a dynamic relationship.

In principle, using *ML Model* a detection algorithm within an ML-based *Detection Pipeline* performs various tasks of detection (e.g. in YOLO8 we can do 'detect', 'segment', 'classify', 'pose' detection [8]). A detection pipeline may employ many detection algorithms. The detection results can be associated with different classification labels (e.g., "person" for an object detection or "spike" for anomaly detection) defined in different naming classes, e.g., classes for objects in YOLO COCO8 [9]. Listing 1 presents an example of an output from Aamzon SageMaker activity detection [10], whereas Listing 2 presents a typical output from a YOLO detection (self implemented).

Hence, in the real-world situations, although the research and industry communities have harmonized such labels, still there exist many naming classes that the downstream applications must handle. A detection result also includes many other types of data, including (i) properties of the detection result such as `"count"` – a specific measurement for `"person"` or

```
%Source code from:
https://github.com/aws-samples/amazon-sagema⌋
ker-activity-detection/blob/master/developme⌋
nt/SM-transferlearning-UCF101-Inference.ipynb
response = {
    'S3Path': {'S': data['S3_VIDEO_PATH']},
    'Predicted': {'S': predicted_name},
    'Probability': {'S': probability},
    'DateCreatedUTC': {'S': now},
}
```

Listing 1: Example from an Amazon Sagemaker code sample [10]

```
result={
    'detection_model_name': model_name,
    'ts': timestamp,
    'result': list_detected_objects
}
```

Listing 2: Example of detection data from a YOLO model

`"anomaly"` – an indicator of an anomaly and (ii) confidence of the detection associated with the detection object via `"Probability"`.

Thus, besides the detection results, there are many other types of data characterizing and explaining the result, e.g. which ML models or configuration are used. Overall, given a CASE, when using multiple ML models for various detection tasks with multiple pipelines, we can consider that the results and associated metadata from these algorithms belong to an "Analytics Feature Space" (AFS), which provides an abstraction for the analytics of CASE. We define an AFS as:

Analytics Feature Space (AFS)

An Analytics Feature Space defines a unified detection management for a collective of assets, spaces, and environments (CASE) at the edge, supporting key metadata, changes, and detection results enrichment to address interoperability, integration, and explainability. A system that supports the AFS concept is called an AFS-aware detection system.

Conceptualizing AFS helps address data interoperability, integration, and explainability of multiple detection algorithms/pipelines for complex applications. First, AFS provides modeling for the results outputted from these tools and their associated metadata in a generic way. This helps improve the selection of detection results based on ML algorithms and their changes in downstream analytics. Second, in order to provide features for advanced analytics, AFS helps the use and composition of the results easily for different applications, from common purposes to specific analytics.

B. Designing AFS

1) Analytics Subject: An *Analytics Subject* is used to represent the subject for detection and analytics. We assume that any AFS-aware detection system will integrate with a CASE management service (e.g., asset management), which have detailed information about the subject. For example, the asset management system can have detailed information about a pump (vendors, profiles, etc.). Analytics subject can represent a CASE. AFS will use the identifier of the subject for discovering associated entities. Every analytics subject is unique in a system (based on naming convention) and for the purpose of integration and explainability, the analytics subject is explicitly model in AFS by tagging an identifier – `analyticsSubjectId`.

2) Detection feature instances: Outputs from an upstream, edge ML detection pipeline are *feature instances* or candidates for *feature instances* to be used in downstream applications (for analytics or further ML tasks), such as safely or runtime operation of all entities the CASE abstracted in an analytics subject in a unified view. Candidates for feature instances will have to be processed through *transformation/enrichment functions* to be *feature instances* for the downstream applications. Therefore, feature instances must also be associated with information about domains and extension for enrichment. Second, the schema for metadata associated with the feature instances is an important problem to be addressed. Such metadata can be collected and enriched through the pipeline.

Currently, many algorithms can detect different types of entities that can be considered for the AFS, such as (i) object detection – types of detected objects are based on a common definition such as person and car [11], [12], (ii) anomaly detection – such as operational problems and faults of assets [13], [14], or behavior detection, such as trends in forecasting. Detailed data fields in the feature instances are specific to application domains. Thus, we can classify the feature instances into different application domains to support the routing and delivery of feature instances through various components in the data pipeline. It is a new problem on how we select, categorize and map the feature instances in a systematically way to suitable for programming, configuring, and developing downstream detection.

3) Explicit naming classes for detection results: Some data fields of feature instances can be improved for data interoperability. Consider a `detected object` in an ML object detection. Although represented in different forms, in most cases, we see that the results from a detection are in the form of *"classname:count"*, illustrated as follows:

```
"result": {
    "person": 3,
    "car": 1
}
```

where *classname* (`"person"` or `"car"`) is the label of the detected object. Furthermore, a confidence degree of the detection can be provided. Given a detection for a data stream, many different ML detection models can be used. Each model is used for different object detection with different accuracy. However, different models may use the same or different class names. For example, `"person"` can be used by two different ML models but it will be an issue to interpret `"persons"`

detected from them as the same, as the "persons" detected are with different accuracy.

Therefore, AFS proposes to map labels/class names from specific ML models to a global name for integration and interoperability purposes. AFS provides an explicit list of naming classes so that classname in the detection results can be traced back to its definitions and detection models. Common prediction classes that can be used in pipelines. We use a common namespace way "namingscheme.classname" to specify information about the scheme where class names are based on. Examples are "UCF101.Archery" ("Archery" is a predicted class in the UCF101 dataset [15]), and "YOLO.Person" (based on YOLO COCO8 [9]).

4) Explicit domain categories for detection purposes: Other important information about the ML detection models used in an existing detection pipeline is *the purpose of the detection*. Such purposes are *implicitly* hidden in the deployment of the pipelines. In terms of sharing for a unified view on detection of a CASE, such results cannot effectively be utilized by other applications without the important information about the purpose.

Understanding the domain and the detection purpose is important. For example, an ML pipeline can detect "persons" but the interpretation is very different if the result is used for traffic and access control management in an industrial zone vs for safety analytics of workers in the same zone. Given a data source, the data captured is for a specific application purpose. Thus, the detection results from that data source are for a specific application purpose, which can be identified by the deployment of the detection pipeline. The purpose can be reflected via the *application domain category*.

Although, the state-of-the art does not have a common agreed vocabulary w.r.t. the category of the application domain, we can assume that they can be pre-defined and user-defined when sharing detection results for CASE. In AFS, we see that the application domain indicates the deployment and goal of detection. We can have a single ML pipeline for a single data source to provide multiple results for multiple applications, which interpret the results differently. In this view, AFS supports a hierarchical domain category and models *the domain category hierarchy* explicitly. This is part of the metadata to associate the results to indicate different application categories. The category hierarchy can be implemented by using hierarchical naming mechanisms.

5) Capturing runtime configuration: Runtime information about an ML-based detection includes two main aspects. First, the ML models used in the pipeline have different runtime parameters, including the composition of multiple models like ensembles [16]. Second, given an ML model and its data sources, the configuration of several parameters, like data sampling rate, window length, and detection frequency, can be customized and configured based on specific underlying deployment. AFS explicitly supports the capture runtime configuration based on a change lifecycle. First, at the deployment time, runtime configuration will be recorded, like a checkpoint, and sent to the downstream. Whenever the configuration changes, AFS-aware pipelines should send a message to inform the change. The message includes a reference to the analytics subject and other changes. Thus, downstream applications will be aware of such changes and make use of the information for their decision.

6) Capturing data source changes: Given the characteristics of an analytics subject explained before, data sources used for the detection are not necessarily fixed in terms of quality and quantity. For example, given a wastewater lake, an existing camera as an data source can be improved if the camera is upgraded with new software or a new data source can be added, such as a new camera is added into a new position. A single detection pipeline using different data sources [17] is very suitable for the analytics of CASE. The information about data source can be modeled based on existing information systems and existing schemas [18]. However, the change of such data sources cannot be detected by the downstream or it is challenging for the downstream to detect the change of data sources by knowing only the detection result. The change is especially important when the detection result may get poorer, such as due to the unavailability of a data source, contributing the quality of input data for the detection.

In AFS, we explicitly model the change of data sources by introducing a schema for sending changes about data sources. A basic lifecycle of a data source is given, covering state like added, removed, unavailable, changed. Similarly to capturing runtime configuration, we can provide APIs for pipelines to checkpoint data sources inside the pipeline (during the deployment and runtime). If there is a change, the change information will be determined and sent. The pipeline can also include quality of data sources that is sampled or observed over the same and send the drift within the change message.

7) Functions for enriching detection results: Feature instances sent to the AFS may not be completed for downstream due to various reasons. First, the detection may decide to put only basic information for efficient purposes, e.g. to be fast or due to limited deployment. Second, at the detection pipeline, information is not available and reference data resides in the place next to the AFS. Therefore, functions can be used to enrich detection results. Such functions can be deployed at various points to perform the enrichment at near real-time.

Another aspect of enrichment is related to the reference sources for metadata. We have to consider the issue that the edge detection deployment is lightweight. Thus, the results may include several references, instead of values. For example, to be efficient, the results from a detection may only include references to ML models, deployment purposes, analytics subject (the subject to be monitored), and data sources. Given such references and sharing purposes, the specific results can be enriched with metadata obtained from reference sources. In some situations, a reference source about the metadata is based on runtime analytics. For example, in oil and gas detection, we may have an hour volume as references for anomaly detection results.

8) Messaging middleware for the AFS (MAFS): Since detection results are mostly delivered at near real-time to the downstream applications, various messaging middleware can be used. In many cases we have to integrate different middleware, due to the diversity of ML detection configurations. We focus on MAFS atop common, well-established middleware (like a combination of MQTT, Apache Pulsar and Apache Kafka) for detection results. We mainly focus on added functions to support the exchanges of AFS-based feature instances and metadata, meaning the routing and enrichment, because techniques and software for bridging and integrating multiple messaging systems are well-developed. Messages encapsulating feature instances are propagated through a scalable middleware. Detection push results of feature candidates/instances to AFS via topics. At the core of AFS, a scalable messaging system will be used for the following main purposes:

- *Context-aware result delivery:* Hierarchical category naming can be mapped to topics/subscriptions. Given the result constraints, detection pipelines can also support the concept of context-aware data sharing by fan-outing or routing the detection results to different topics. This allows dynamic data sharing at the pipelines and MAFS level, rather than at the data hub seen in existing works.
- *Integration of functions for enriching results*: the integration is based on two different deployment modes: the internal deployment mode explores serverless and streaming analytics of the messaging brokers (e.g., Pulsar Functions [19] and kSQL [20]) to deploy functions. The external model deployment allows functions to consume messages that can be implemented using docker-based microservices or severless functions.
- *Delivery of additional non-message metadata:* Additional, non message-based data, such as an image, short clip or a CSV dataset, linked to feature instances can also be supported. They can be delivered to downstream systems via two mechanisms: pull from an edge storage or push to a cloud based storage.

9) Integration with pipeline development: Metadata and results as well as AFS models can be used differently: (i) the implementation of pipelines do not use AFS but the downstream applications use AFS to transform non-AFS results. (ii) existing pipelines can be modified to add metadata into detection results, and (iii) the implementation of pipelines completely uses new schemas from AFS. A light pipeline with limited ML models may detect one types of objects, thus its feature instances can be associated with a single a category or domain. A complex pipeline may detect many things, thus the results include several types of detection, each can have different naming classes.

IV. Implementation

Based on the concept of AFS presented in Section III, in this section we present one implementation of the AFS.

1) Hierarchical category: Figure 3 gives a snapshot of the hierarchical category for the application purposes. The principle is that they can be used to annotate the domain with the detection results. AFS does not enforce a common one and they can be defined based on the specific deployment.

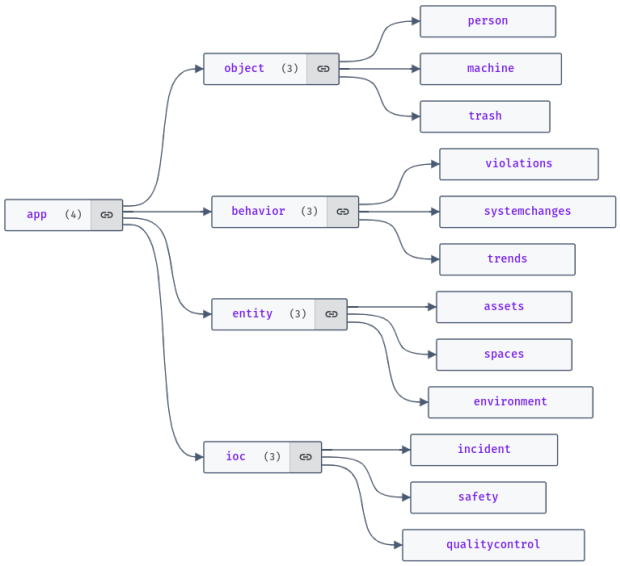

Fig. 3. Example of the hierachical category. Note that these categories are customized based on applications and deployments.

2) Detection configuration change: List 3 shows the basic model for making/informing the change of the detection pipeline: e.g., window, sampling, topology, detection model, that will be recorded by using a checkpoint API. Given a `timestamp` and `analyticsSubject` we know that the configuration has been changed.

```
class DetectionConfig(BaseModel):
    timestamp: float
    analyticsSubject: AnalyticsSubject = None
    domainCategory: str = None
    detectionModel: DetectionModel = None
    config: dict = None
```

Listing 3: Changes in runtime configuration.

3) Data source change: Listing 4 presents the schema for capturing and sending the change. We use existing data observability tools for sampling and profiling data from the sources, such as Ydata [21]. At the deployment of the pipeline a checkpoint will write the data sources and when there is change the system will have to do this. This is not carried out automatically but the API has to be instrumented by the developer of the ML pipeline.

4) Feature Instance Schema: Features provided by an ML detection are specific to the ML pipeline implementation. There are different ways to integrate AFS models with current results from existing pipelines by using AFS schemas and functions. Listing 5 presents AFS Feature Instances and a common structure to encapsulate features from specific pipelines. However, each pipeline can have its feature structure and use AFS metadata.

```
class DataSource(BaseModel):
    name: str=None
    type: str=None
    sourceRefId: str=None
    properties: dict=None

class DataSourceChange(BaseModel):
    timestamp: float
    analyticsSubject: AnalyticsSubject = None
    changeType: ChangeTypeClassEnum = None
    dataSources: List[DataSource]
```

Listing 4: Modeling changes in data sources.

```
class CommonDetectionResult(BaseModel):
    domainCategory: str = None
    detectionResult:
    Union[ObjectDetectionResult,dict]
    detectionSourceRefId: str = None
    additionalData: dict = None

class FeatureInstance(BaseModel):
    featureInstanceId: str
    timestamp: float
    detectionResult:
    List[CommonDetectionResult] = None
    analyticsSubject: AnalyticsSubject
    domainCategory: str = None
    detectionModel: DetectionModel = None
    detectionConfig: DetectionConfig = None
```

Listing 5: A schema for representing feature instances.

5) Common result handlers for detection pipelines: Results from a detection ML are sent to middleware for downstream applications. In cases of specific results (no AFS), a handler may take the specific result and send it. In cases of a AFS-aware system, the handler behavior has to be changed by (i) transforming the specific result to AFS and (ii) filtering/routing results based on the classname and domain of AFS. In our implementation, we provide baseclass for `ResultHandler`. Each handler can be configured with a data transformation to send data into the space or customers. Listing 6 shows two basic APIs for supporting edge pipeline. Both handlers and transformations for handlers can be dynamically loaded using configuration information. When sending data into AFS, it is also possible to use the domain category or class name to configure the topic and channel (e.g., for publish and subscription models).

6) Storing and Messaging: Detection results are delivered by using MQTT, Pulsar and Kafka. Changes are also delivered as messages but they are different so that the downstream applications can distinguish between detection results and changes. However, for some data required for the references to the detection, such as images and detail logs, we need to use a different way. In this case, AFS supports REST-based APIs that pipelines can use to push the data and provide references into messages.

7) AFS Enrichment Functions: AFS functions are implemented differently from transformation/handlers used at the

```
#for processing and pushing results to
messaging systems and target services
@abstractmethod
def __process__(self,result):
   pass
#for transforming specific results to
AFS-aware results
@staticmethod
def transform(input_detection_result,filtere
d_domain_category_instance=None,filtered_dom
ain_category_result=None,filtered_classname=
None):
    pass
```

Listing 6: Common abstract APIs for sending and transforming features at the pipeline side.

edge pipeline side to send results to the space. We currently implement two mechanisms: function embedded into the messaging system and external docker containers. For embedded function, it is dependent on the messaging system. For example, shown in Figure 4 this can be done by having a task in the streaming processing, e.g., with kSQL [20] or serverless function like Pulsar functions [19]. The external functions in containers are well understood and follow a typical model of bridges of consuming, transforming and republishing data.

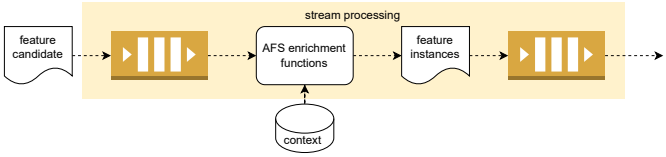

Fig. 4. AFS enrichment functions deployed within messaging systems.

V. Experiments

A. Testbed and experiment settings

As we cannot present experiments with a real IOC setting, we emulate IOC scenarios and their CASE using existing data sources and common ML pipelines. Figure 5 illustrate the testbed:

- ML-based object detection is for detecting people in buildings and specific spaces (such as surrounding wastewater lakes).
- ML-based sound detection (using data from [22]) is for detecting anomaly of assets (such as the pumps).
- ML-based anomaly detection is for detecting the environmental quality of the wastewater (such as conductivity, pH, etc. using data from [23]).

These detection pipelines have been developed and deployed differently and then integrated using MQTT/Pulsar/Kafka and RQ Queue [24] to push data to downstream applications. Other external services for the testbed are integrated:

- *ML Model Registry*: We use MLflow [25] to store detailed ML model information for enriching detection models, such as YOLO models for object detection, as well

as ML benchmark details of other real-world models for object classification[2], including accuracy, throughput, risks associated with classification, etc. In addition, we also store basic ML models for anomaly detection of industrial audio signals trained using data from [22]) and of time-series sensor measurements.

- *Analytics Subject Management*: a service based on MongoDB is used for storing details about analytics subjects and assets. The asset information is based on our experiences with industrial asset management.

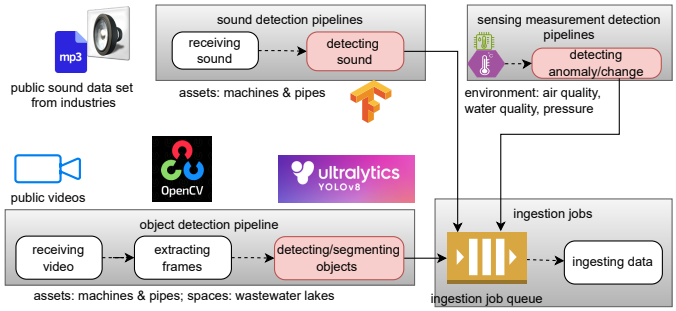

Fig. 5. Emulation testbed.

For the downstream application, we consider a situation in which the IOC deploys operation management. When a problem happens in a CASE, a collection of related anomalies of assets and environments at a given time needs to be extracted for additional information to deal with the problem or to find the root cause. The object detection can also support the analytics if it signals some important information, such as people in the space or assets that can cause some problems. When accessing the metadata supported by AFS, all anomalies and detected objects related to the CASE can be easily identified in a unified way.

B. Examples of using AFS models and functions

Let us consider a factory in an IOC `ioc-f01` and a CASE `f01-case01`. Three entities in `f01-case01` are a pump `pump01` (an asset), a wastewater lake `wastewater01` (an environment), and the space around the wastewater lake `lakespace01` (a space). Within this CASE, we define a single analytics subject representing three entities `as-case01wastewater` which aims at demonstrating the management of the collective, instead of individual entities.

In the following, different ways of using AFS are emphasized. Listing 7 illustrates the output data given by the specific schema of an ML pipeline for object detection for entity `lakespace01`. Such a detection result requires a deep integration between the service handling the detection result and the upstream ML detection. For example, the detection result classes can be interpreted well only if the downstream application knows the information about the ML model used. To be interoperable, one solution is that the data based on this

[2]obtained from a national research institute whose name is withdrawn for double blind submission

```
{
    "detection_model_name": "yolov8n.pt",
    "ts": 1710591222.862264,
    "results": {
        "person": 3
    },
    "source_image_uri":
    "source_1710591222.862264.jpg",
    "annotated_image_uri":
    "annotated_1710591222.862264.jpg"
}
```

Listing 7: An object detection record based on a specific schema.

schema can be replaced by using AFS schema (implemented within the pipeline) or can be transformed into AFS schema (using an adaptor or a function). Listing 8 presents an example AFS-based data that can be obtained either by transforming the specific schema or using AFS API in detection pipelines. During runtime, the upstream detection can decide changing the result constraints that affect the detection results, e.g., when there is a human work at `lakespace01`. Listing 9 shows an example of the change message of the detection constraints. Similarly, changes of data sources associated with a pipeline can also be sent. Details about an ML model used in a pipeline can be enriched by using `ML Model Registry` and the name of the model encapsulated in the detection result, e.g. shown in Listing 10. Similarly, detailed about entities in an analytics subject can be enriched by calling the *Analytics Subject Management* service.

C. Example of downstream analytics applications

For newly-developed ML pipelines, detection results can be represented by using AFS schemas and other enrichment can be done by functions in the corresponding AFS middleware. However, without changing anything, detection results from an existing upstream pipeline can be transformed and enriched with AFS metadata at the downstream. Figure 6 shows a configuration for the operation management application discussed in Section V-A, in which results based on specific ML schemas will be transformed into AFS schema. In this configuration, many detection pipelines are deployed for streaming data sources. Different types of pipelines have different types of results, e.g., for classes and numbers of detected objects (`lakespace01`), anomalies from audio signals (`pump01`), anomalies from sensing measurements (`wastewater01`), and target for different applications (safety vs quality control). These above-mentioned results are managed uniformly for all downstream applications using a file storage (Minio), a document database (MongoDB), and a vector database (Milvus). However, individual detection results are streaming for different applications. Then applications can decide to subscribe suitable topics to get detection results in the same format (but different types).

For the operation management application, when an incident or problem happens in the analytics subject

```
{
    "featureInstanceId":
    "956947ff-a05f-492d-a8fc-669d4880653a",
    "timestamp": 1710591222.862264,
    "detectionResults": [
        {
            "detectionResult": {
                "classname": "yolo_categorym
                apping.objects.person",
                "count": 3
            },
            "detectionSourceRefId":
            "as-case01wastewater_533a1f0e-20
            f2-4f15-ace8-f31baae1d7e1_source
            _1710591222.862264.jpg",
            "additionalData": {
                "ref_doc": "as-case01wastewa
                ter_533a1f0e-20f2-4f15-ace8-
                f31baae1d7e1_annotated_17105
                91222.862264.jpg"
            }
        }
    ],
    "analyticsSubject": {
        "analyticsSubjectId":
        "as-case01wastewater"
    },
    "domainCategory": "app.ioc",
    "detectionModel": {
        "detectionModelName": "yolov8n.pt"
    }
}
```


Listing 8: Example of transform the specific one to the AFS model. The detection results can include many records.


```
{
    "timestamp": 1710591220.340439,
    "analyticsSubject": {
        "analyticsSubjectId":
        "as-case01wastewater"
    },
    "detectionModel": {
        "detectionModelName": "yolov8n.pt"
    },
    "config": {
        "result_constraints": {
            "person": {
                "operator": ">",
                "value": 2
            }
        }
    }
}
```


Listing 9: Example of a changes of result constraints.


```
{
    "detectionModelRefId": "pump_id_04_6dB",
    "detectionModelName": "pump_id_04_6dB",
    "modelInfo": {
        "details": {
            "name": "pump_id_04_6dB",
            "last_updated_timestamp":
            1710583057537,
            "tags": [
                {
                    "key":
                    "detectionModelRefId",
                    "value": "pump_id_04_6dB"
                },
                {
                    "key": "detectionType",
                    "value":
                    "anomaly_detection"
                },
                {
                    "key": "AUC",
                    "value": "0.987"
                }
            ]
        }
    }
}
```


Listing 10: Example of enriching ML model information.

`as-case01wastewater`, further analytics are required to check the cause and evidence:

- Operators need to identify all entities in `as-case01wastewater`. This can be easily done by using CASE management integrated with AFS-aware features.
- After that, operators can use the reference and documents from the operation management service to obtain images and all anomalies of identified entities that would be related to the incident or problem.

Listing 11 illustrates a simple code excerpt triggering the search for relevant evidence. First, the document database can be searched to find relevant detection results of the object detection within a window of time (based on the time when anomalies in audio signals and sensing measurements are detected). When checking the detection result returned as shown in Listing 12, a further search, e.g., similar search, can be done by retrieving the evidence (`ref_image`) of the detection result from the file storage.

VI. Related Work

Edge ML detection: A large number of works on object detection at the edge in various scenarios have been published [1], [2], [3], [4]. They many cover (i) ML models and algorithms for detection, (ii) systems for integration of detection and (iii) performance evaluation. In the industry, there are many tools and models available, like [8]. However, the major problems that can be seen when utilizing them are specific data models and lack of metadata. Another problem is that these works do not address the detection and communication of changes

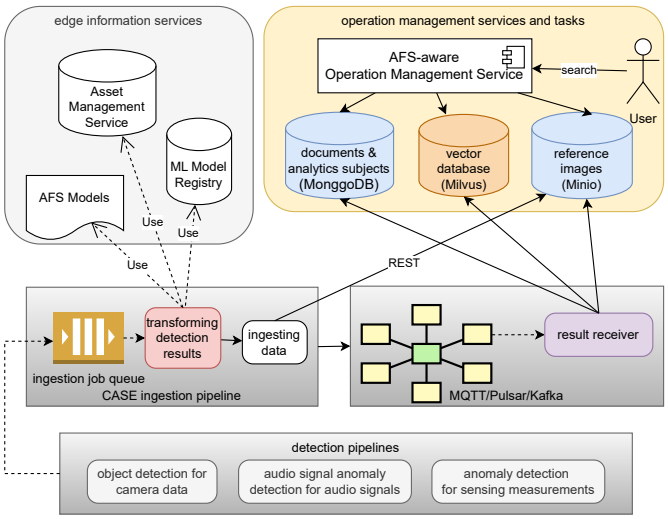

Fig. 6. Example of configuration and integration for operation management downstream application use case.

```
analytics_subject="as-case01wastewater"
#...
start_time = specified_time - interval/2
end_time = specified_time + interval/2
query = {
    "timestamp": {'$gte': start_time, '$lt':
    end_time},
    "analyticsSubject.analyticsSubjectId":
    analytics_subject
}
results = db_collection.find(query,projectio
n=projection)
for result in results:
    #...
#get image/evidence in the select result:
temp_data_file=os.path.basename(selected_res
ult["ref_image"])
#get the image
minio_utils.downloadtolocal(bucket,selected_
result["ref_image"],temp_data_file)
print(f'Evidence image: {temp_data_file}')
```

Listing 11: Simple code illustrating the query of relevant evidences when a streaming detecting anomalies in both `wastewater01` and `pump01`.

in detection deployment to the downstream. When utilizing single models in a single system, one does not face the interoperability problem. However, it is a great challenge for the developer when dealing with different detection pipelines. Our work focuses on reducing such challenges.

Metadata and data transformation: One solution is to use different transformation functions to transform the specific data into a common model. However, such a solution does not provide an interoperable structure, meaning that the choice of transformation and target are specific. Work in [26] focuses on querying data to support adaptation. This is a kind of "downstream" applications based on multiple sources of data. Detection results with metadata could help to improve such a

```
{
    "id":
    "c53f2ba0-921c-4b4b-adf2-227f6f6b8a0a",
    "details": {
        "featureInstanceId": "c53f2ba0-921c-
        4b4b-adf2-227f6f6b8a0a",
        "timestamp": 1710616161.826744,
        "detectionResults": [
        ...
        ]
    },
    ...
    "ref_image": "as-case01wastewater/as-cas
    e01wastewater_8996d661-b4f0-492e-8bd4-b8
    7f612a2732_source_1710616161.826744.jpg"
}
```

Listing 12: Examples of object detection results for `lakespace01` when the anomalies occurred at `pump01` and `wastewater01`.

work. The paper [27] focuses on metadata and data modeling for assets and processes. Our work differs as we concentrate on the integration and exchanges of detection results and analytics.

VII. Conclusions and Future Work

If tightly integrated or implemented and controlled by a single vendor in a close edge-to-cloud infrastructure, the results from multiple detection pipelines might not need much metadata. However, today and future's ML detection relies on multiple ML models and pipelines developed by different vendors, is based on marketplace ML models and blackbox ML as a service [28], [29], and is highly decoupling in terms of operations and management responsibility. Thus, addressing metadata for the integration of multiple types of edge ML detection pipelines are of paramount importance. We have introduced AFS as a novel framework to abstract individual detection and unified them for analytics of subjects associated with collectives of assets, spaces and environments. AFS introduces various schemas and techniques to link and enrich metadata for detection integration and interoperability.

Our future work is to improve the prototype and integrate AFS into real-world systems of intelligent operation centers. Furthermore, we will focus on the enrichment and downstream applications, which will help to improve and extend designs of AFS.

References

[1] M. G. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *ACM Comput. Surv.*, vol. 54, no. 8, oct 2021. [Online]. Available: https://doi.org/10.1145/3469029

[2] S. Nagaraj, B. Muthiyan, S. Ravi, V. Menezes, K. Kapoor, and H. Jeon, "Edge-based street object detection," in *2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, 2017, pp. 1–4.

[3] L. U. Khan, I. Yaqoob, N. H. Tran, S. M. A. Kazmi, T. N. Dang, and C. S. Hong, "Edge-computing-enabled smart cities: A comprehensive survey," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10 200–10 232, 2020.

[4] Y.-Y. Chen, Y.-H. Lin, Y.-C. Hu, C.-H. Hsia, Y.-A. Lian, and S.-Y. Jhong, "Distributed real-time object detection based on edge-cloud collaboration for smart video surveillance applications," *IEEE Access*, vol. 10, pp. 93 745–93 759, 2022.

[5] S. Mehnaz and E. Bertino, "Privacy-preserving real-time anomaly detection using edge computing," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 2020, pp. 469–480.

[6] M. V. Ngo, T. Luo, and T. Q. S. Quek, "Adaptive anomaly detection for internet of things in hierarchical edge computing: A contextual-bandit approach," *ACM Trans. Internet Things*, vol. 3, no. 1, oct 2021. [Online]. Available: https://doi.org/10.1145/3480172

[7] H. Xiang and X. Zhang, "Edge computing empowered anomaly detection framework with dynamic insertion and deletion schemes on data streams," *World Wide Web*, vol. 25, no. 5, pp. 2163–2183, 2022. [Online]. Available: https://doi.org/10.1007/s11280-022-01052-z

[8] "Ultralytics yolov8 tasks," last access: Mar 9, 2024. [Online]. Available: https://docs.ultralytics.com/tasks/#detection

[9] "Ultralytics yolo – coco8 dataset," last accessed: Mar 27, 2024. [Online]. Available: https://github.com/ultralytics/ultralytics/blob/main/ultralytics/cfg/datasets/coco8.yaml

[10] "Transfer learning and action inference on input video segments," last accessed: Mar 27, 2024. [Online]. Available: https://github.com/aws-samples/amazon-sagemaker-activity-detection/blob/master/development/SM-transferlearning-UCF101-Inference.ipynb

[11] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, "A survey of modern deep learning based object detection models," *Digital Signal Processing*, vol. 126, p. 103514, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1051200422001312

[12] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *Proceedings of the IEEE*, vol. 111, no. 3, pp. 257–276, 2023.

[13] A.-A. Tulbure, A.-A. Tulbure, and E.-H. Dulf, "A review on modern defect detection models using dcnns – deep convolutional neural networks," *Journal of Advanced Research*, vol. 35, pp. 33–48, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2090123221000643

[14] E. Westphal and H. Seitz, "A machine learning method for defect detection and visualization in selective laser sintering based on convolutional neural networks," *Additive Manufacturing*, vol. 41, p. 101965, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214860421001305

[15] "Ucf101 – action recognition data set," last accessed: Mar 27, 2024. [Online]. Available: https://www.crcv.ucf.edu/data/UCF101.php

[16] A. D. Blaom and S. J. Vollmer, "Flexible model composition in machine learning and its implementation in MLJ," *CoRR*, vol. abs/2012.15505, 2020. [Online]. Available: https://arxiv.org/abs/2012.15505

[17] "Robotflow – multiple video sources," last accessed: Mar 27, 2024. [Online]. Available: https://github.com/roboflow/inference/releases/tag/v0.9.18

[18] N. W. Paton, J. Chen, and Z. Wu, "Dataset discovery and exploration: A survey," *ACM Comput. Surv.*, vol. 56, no. 4, nov 2023. [Online]. Available: https://doi.org/10.1145/3626521

[19] "Pulsar functions overview," last accessed: Mar 27, 2024. [Online]. Available: https://pulsar.apache.org/docs/next/functions-overview/

[20] "Introducing ksql: Streaming sql for apache kafka," last accessed: Mar 27, 2024. [Online]. Available: https://www.confluent.io/blog/ksql-streaming-sql-for-apache-kafka/

[21] YdataAI, "YData-quality data quality assessment with one line of code," 2023, last accessed: Mar 09, 2024. [Online]. Available: https://github.com/ydataai/ydata-quality

[22] H. Purohit, R. Tanabe, K. Ichige, T. Endo, Y. Nikaido, K. Suefusa, and Y. Kawaguchi, "MIMII dataset: Sound dataset for malfunctioning industrial machine investigation and inspection," *CoRR*, vol. abs/1909.09347, 2019. [Online]. Available: http://arxiv.org/abs/1909.09347

[23] S. Lindgren and B. d. Bruin, "Water quality data from talkpool sensor," Apr. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7860472

[24] "Rq (redis queue)," last accessed: Mar 16, 2024. [Online]. Available: https://python-rq.org/

[25] "Mlflow model registry," last accessed: Mar 12, 2024. [Online]. Available: https://mlflow.org/docs/latest/index.html

[26] J. de Oliveira, C. Calle, P. Calvez, and O. Curé, "Towards autonomous anomaly management using semantic technologies at the edge," in *2023 IEEE International Conference on Edge Computing and Communications (EDGE)*, 2023, pp. 159–165.

[27] H. Shi, S. Liu, and L. Pan, "Domain modeling for scenario sensing and edge decision-making," in *2023 IEEE International Conference on Edge Computing and Communications (EDGE)*, 2023, pp. 118–125.

[28] J. Pei, R. C. Fernandez, and X. Yu, "Data and ai model markets: Opportunities for data and model sharing, discovery, and integration," *Proc. VLDB Endow.*, vol. 16, no. 12, p. 3872–3873, aug 2023. [Online]. Available: https://doi.org/10.14778/3611540.3611573

[29] J. Liu, J. Lou, J. Liu, L. Xiong, J. Pei, and J. Sun, "Dealer: an end-to-end model marketplace with differential privacy," *Proc. VLDB Endow.*, vol. 14, no. 6, p. 957–969, feb 2021. [Online]. Available: https://doi.org/10.14778/3447689.3447700