Video Situation Monitoring Using Continuous Queries

Hafsa Billah(✉) and Sharma Chakravarthy

IT Lab and CSE Department, University of Texas at Arlington, Arlington, TX, USA
uxb7123@mavs.uta.edu, sharmac@cse.uta.edu
https://itlab.uta.edu/

Abstract. Video situation monitoring is important for many applications such as infrastructure surveillance, traffic monitoring, etc. Currently, situations are monitored either manually using human-in-the-loop or custom algorithms. Manual approach was applicable for short videos. Monitoring situations in hours of long videos manually is difficult and subject to human error. On the other hand, custom algorithms are designed for specific situations and video types. A new algorithm or software package must be written for every new situation type. Both of the above approaches cannot monitor situations automatically. In this paper, we propose an alternative to the above two approaches to facilitate automated situation monitoring by posing situations as queries. The proposed approach minimizes or avoids human involvement and avoids writing new software packages or algorithms for every new situation type. The proposed framework extracts video contents *only once* using existing video content extraction algorithms. Appropriate data models and new operators and algorithms for efficient situation analysis are required to perform *ad-hoc* and *what if* querying on the extracted contents for situation monitoring. This paper extends the traditional relational model with support for representing various extracted content types. The Continuous Query Language (CQL) is also extended with new operators for posing situations as continuous queries. Backward compatibility, ease of use, primitive new operators (including spatial and temporal), and algorithms for efficient execution are discussed in this paper. Finally, query correctness with manual ground truth, efficiency, and the robustness of the algorithms are demonstrated.

Keywords: Video content extraction · Situation monitoring · Continuous queries

1 Introduction

Video data is being generated daily in large volume and variety because of the availability of inexpensive camera devices (e.g., personal cameras, CCTV, etc.). Analyzing these videos is important for different applications such as traffic monitoring, individual infrastructure surveillance (e.g., parking lot, shopping mall, etc.), postmortem analysis of criminal activities, and many others. For example, some of the suspects in the Capitol riot incident have not been apprehended yet. An automated video situation analysis system would have sped up the search for the suspects by warranting the law enforcement

C. Strauss et al. (Eds.): DEXA 2024, LNCS 14911, pp. 125–141, 2024.
https://doi.org/10.1007/978-3-031-68312-1_10

officials when these suspects were seen in the CCTV footage of any important places (e.g., grocery store, parking lot, etc.) Currently, there are two approaches to situation analysis. They are manual analysis and custom solutions. In the manual analysis process, a human watches a video exhaustively and looks for situations of interest. This is labor intensive, subject to human error, and infeasible for large videos. Custom analysis algorithms are designed for specific situations or video types. A new algorithm needs to be written for each new situation type. Even with several custom algorithms available, the capitol riot incidents were not analyzed. This paper takes an alternative approach to the above two approaches, which can analyze situations continuously from video streams by posing situations as continuous queries. The proposed approach can avoid or minimize human-in-the-loop and eliminate the need for writing custom solutions. This solution can be eventually extended to a real-time situation monitoring system, which cannot be achieved using the above two approaches.

Video Content Extraction (VCE) is a prerequisite for any kind of automated situation analysis task [17]. Meaningful contents (e.g., object location as a bounding box, object attributes as feature vectors, etc.) can be extracted using advanced deep learning algorithms. These extracted contents can be streamed to a sensor Stream Processing **(SP)** framework for continuously querying video streams (and analyzing situations from). However, for several reasons, existing sensor SP framework functionalities cannot be directly applied to extracted video contents. First, the extracted video contents are very different from sensor data as they contain multi-dimensional feature vectors, bounding boxes, etc. These extracted contents cannot be modeled using the existing relational model used in sensor SP frameworks. Second, extracted content analysis for situation monitoring requires different computations (e.g., comparing multi-dimensional feature vectors based on similarity) and traditional relational operators are not sufficient for these computations. An appropriate representation model with new operators for processing the extracted contents is needed to analyze the relevant situations. In this paper, we propose that video situation analysis can be effectively enhanced and automated by *extending* and synergistically integrating approaches from VCE and SP for obtaining an end-to-end holistic solution. The **contributions** of this paper are:

– A non-traditional low-risk framework for **Querying Video Contents (QVC)** and its viability for video situation monitoring (Sect. 4).
– An expressive extended relational model (**R++**) that supports variety of extracted video content types (Sect. 4).
– A **Continuous Query Language for Video Analysis (CQL-VA)** with new operators and algorithms for processing extracted video contents (Sect. 5).
– **CQL-VA query formulation** for primitive situation analysis (Sect. 6) and **Experimentation** for accuracy, efficiency, robustness, and scalability (Sect. 7).

The challenges, related work, and conclusion are in Sect. 2, 3, and 8 respectively.

2 Challenges

Robust situation detection is dependent upon what contents can be extracted by the VCE algorithms. The immediate goal of this paper is to formulate queries that can be posed and processed on the extracted video contents. In Table 1, we have summarized a list of primitive situations that can be analyzed using the extracted contents. These situations are a starting point for video situation analysis involving aggregation/boolean queries, where information from the video is aggregated along temporal or spatial dimensions. Once these primitive situations are addressed, they can be composed further to answer more complex situations such as "Has the same person entered or exited in a video more than 'n' times?".

Problem Statement: *Develop a framework to a) extract video contents once, b) represent the extracted contents using an expressive data model, and c) design primitive operators to answer queries (both ad-hoc and "what if") on the extracted contents.*

Table 1. Example Primitive Situations

Query	Category	Example Situations	Computation
Q1	Searching	Searching for an object in a video using its image	Spatial
Q2	Aggregation	Busy period in a public area (e.g., shopping mall)	Temporal
Q3	Join	Is the same person present in two or more videos (captured using an entry or exit camera)?	Temporal
Q4	Direction	Which direction an object moved in the video?	Temporal

There are several challenges in addressing the situations in Table 1. First, **extracted video contents cannot be modeled** completely with the existing relational model. For example, Q4 in Table 1 requires computing the direction between two bounding boxes. Modeling an object bounding box using a traditional relational model would require four columns. A self-join is required to bring the bounding boxes of two frames for comparison using the relational model. This will be much more difficult when multi-dimensional feature vectors need to be compared for a situation (e.g., Q1, Q3 in Table 1). Structured Query Language (SQL) and Procedural Language extensions to the SQL (PL/SQL) have provisions for representing multi-dimensional vectors using JSON array or VARRAY data type, respectively. Performing computations and expressing queries using these arrays are complicated. The array DBMSs [5] also support multi-dimensional vectors of one type (e.g., coordinate positions), and the whole relation is modeled as a multi-dimensional vector. Even though feature vectors can be represented using this model, other extracted content types (e.g., object class label) cannot be represented. Though the column-oriented semantics proposed in AQuery [11] can represent two-dimensional array, they cannot model multi-dimensional feature vectors. Computations on the different extracted content types are also not supported by the above representation models. To answer the situations in Table 1 using SQL, PL/SQL, array DBMSs, or object-oriented databases would require writing user-defined functions for each of them. This is similar to developing a custom algorithm for each situation.

Performing joins on videos is another challenge. Traditional joins compute matches by applying relational model comparison operators. These operators cannot compare multi-dimensional feature vectors, as they vary significantly for the same object and must be compared based on similarity. Hence, A matching condition (with an appropriate similarity measure) is needed. Consecutive frames in a video contain repetitive information (e.g., the same set of objects in multiple frames). Not all frames are necessary for many computations. For example, counting an object occurrence once across frames is sufficient to answer Q2 in Table 1. For Q3, all the frames from two videos are not required to be compared. Hence, new operators are needed to compress the repetitive information and efficiently process it, which is not supported by existing systems.

3 Relevant Work

Different aspects of video content analysis have been researched for a long time. We have summarized these works into three different categories in Table 2. They are Custom Solutions (CS), Video Streaming (VS) systems, and Low-level Content Analysis (LCA) systems. We have also shown in Table 2 whether these systems address the primitive situations from Table 1. CS shown in Table 2, row 1, are mostly deep learning approaches having fixed situation classes, such as searching for a person or spatial relationship between objects in a video. Their main focus is neural network optimization (speeding up training and inference), and they need to be retrained for a new situation type. For example, NoScope [10], MIRIS [4], and SVQ++ [6] can address partially Q1 from Table 1. They cannot address the situations Q2-Q4 from Table 1 and compose the situations to address complex situations. Existing VS systems (row 2 in Table 2) support storing, searching, and retrieval of video/image based on some metadata. Though they are efficient for streaming video frames, there is no query processing support.

Table 2. Summary of existing video content analysis literature vs. QVC.

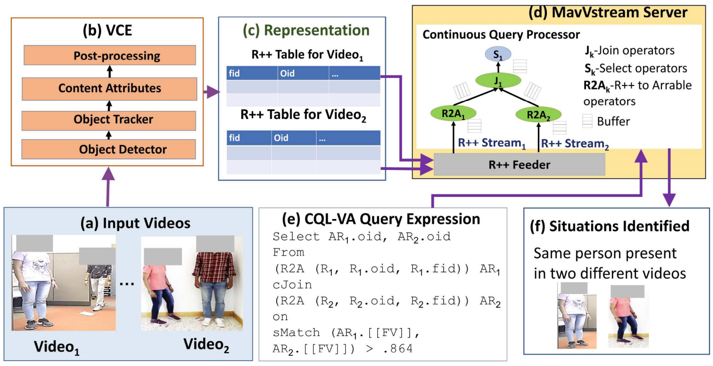
Category	System	Data Model	Supported Situations			
			Q1	Q2	Q3	Q4
CS	NoScope [10], MIRIS [4], SVQ++ [6]	Video	Partial	No	No	No
	BLAZEIT [9]	Relational	No	No	No	No
VS Systems	Amazon Kinesis [13]	Relational	No	No	No	No
LCA Systems	BilVideo [7], SVQL [12]	Relational	No	Yes	No	Yes
	LVDBMS [3]		No	Yes	No	No
Extended SP	Proposed QVC Framework	R++	**Yes**	**Yes**	**Yes**	**Yes**

The LCA systems (row 3 in Table 2) model video contents using the relational model and support several low-level spatial operators (bounding box overlap, direction, etc.). BilVideo [7] and SVQL [12] allow query processing on fixed event and content databases. These databases must be updated frequently, as extracted video contents and situations (or events) change over time. LVDBMS [3] also supports some spatial operations. However, they do not support window-based joins or other window-based operations. Even though these systems can answer aggregation queries (Q2 from Table 1), they do not have the functionality to answer Q1, Q3–Q4 from Table 1. Recently, graph-based analysis [16–18] has become popular for video situation monitoring. However, they can not also answer Q1, Q3–Q4 from Table 1. They are not discussed here elaborately, as this paper extends the relational model.

4 Proposed QVC Framework

The proposed QVC framework (shown in Fig. 1(b–d)) is composed of three modules: A) Video Content Extraction (VCE), B) Representation of extracted contents using extended relational (R++) model, and C) Continuous query processing with CQL-VA operators for situation detection. These components are discussed below.

A) Video Content Extraction (VCE): The VCE component of the proposed framework employs two VCE algorithms: object detection (YOLO [14]) and object tracking (deep sort [15]). Once all the content attributes are extracted, they are post-processed for appropriate formatting. After post-processing, the VCE module extracts **Frame id (fid)**: a unique identifier of a frame, **Object id (oid)**: a unique identifier for an object across frames assigned by object tracking, **Object class label**, **class confidence score**, **Bounding box ([BB])**: object location in a particular frame, **Feature vector ([FV])**: multi-dimensional vectors representing an object feature, and **timestamp (ts)**: a timestamp for each 'f' frames, where f is the frame-per-second rate of the video.



```
Select AR1.oid, AR2.oid
From
(R2A (R1, R1.oid, R1.fid)) AR1
cJoin
(R2A (R2, R2.oid, R2.fid)) AR2
on
sMatch (AR1.[[FV]],
AR2.[[FV]]) > .864
```

Fig. 1. Video situation analysis steps: (a) Input video streams, (b) VCE with post-processing, (c) Representation (using R++ model), (d) Continuous Query Processing (using MavVStream server), (e) CQL-VA Query expression submitted by the user, and (f) Situations identified.

B) Representation of Extracted Contents: The VCE module discussed above extracts three different categories of contents: numerical (fid, oid, ts), enumerated (object class label and confidence score), and vectors ([FV], [BB]). These contents cannot be processed in an arbitrary order. For example, to answer Q4 in Table 1, the direction of an object's bounding box in the first and last frame it appears must be computed. Besides, a self-join is required to compare all four elements of two bounding boxes to answer

Q4 using the relational model. It would have been much simpler to express and avoid self-join if all the information associated with an object across frames could be represented with one tuple. The column-oriented semantics of the arrable data model and the order-preserving operations in AQuery [11] supports the above to some extent.

An arrable is an ordered collection of vectors (or arrays) of basic type (e.g., numeric, boolean, etc.). The arrable data model represents the tuples in a partition with one tuple after grouping and ordering on a set of attributes. In other words, each group is represented with one tuple using arrable data model. In Fig. 2(c), R_1 was grouped on oid and ordered on fid to generate AR_1. All the attributes of AR_1 are arrable except the group by attribute oid. As mentioned in Sect. 2, the arrable data model cannot model multidimensional feature vectors and other different content attributes extracted by VCE. Hence, the proposed extended relational model (termed R++) supports three different data types. They are basic types (numerical, enumerated for representing object class labels, directions, etc.) from the relational model, vectors (n-dimensional), and **extended arrables**. An R++ relation and extended arrables are defined below.

Definition 1. *An R++ relation R consists of attributes $A_1, A_2, ..., A_n$. An attribute A_i can be of basic, vector, or an extended arrable type. An extended arrable attribute A_i is an n-dimensional array generated by grouping and ordering on the numeric or enumerated attributes of R.*

A tuple is inserted in an R++ relation for each object extracted from a video frame with its associated attributes. In other words, for each frame processed by VCE, k rows are added incrementally to the R++ relation corresponding to a video, where k is the number of objects extracted by VCE. An example of an R++ table is shown in Fig. 2(b). Here, five different types of object attributes are shown. fid, oid, and ts are numerical attributes, whereas [BB] and [FV] are multi-dimensional vector types. [BB] vector size is four, and [FV] vector size varies depending on the VCE algorithm. In Fig. 2(b), frame 1 contains two objects with oid 1 and 2. Therefore, there are two tuples for frame 1 (with different oids) in the relation. To support a time-based window, each frame is associated with an actual timestamp (shown as an integer for convenience). Here, feature vectors are shown as one-dimensional vectors for simplicity.

An example of an R++ table containing extended arrables is shown in Fig. 2(c). This table contains one tuple per oid, instead of containing a tuple for each fid and oid pair (see Fig. 2(b)). This allows us to represent all the [BB], [FV], ts, fid, etc., associated with an oid from R_1 using an array across the entire relation (or even a window, which is a horizontal partitioning using ts). All the attributes of AR_1

(a) Input video

(b) An R++ table, R_1

fid	oid	[BB]	[FV]	ts
1	1	[10,20,4,5]	[1,2,.7,1.1]	1
1	2	[15,40,7,8]	[.5,.4,.3,.4]	1
2	2	[30,55,7,8]	[.9,.4,.3,.45]	2

(c) An R++ table with extended arrables, AR_1 = R2A (R_1, oid, ts)

[fid]	oid	[[BB]]	[[FV]]	[ts]
[1]	1	[[10,20,4,5]]	[[1,2,.7,1.1]]	[1]
[1,2]	2	[[15,40,7,8], [30,55,7,8]]	[[.5,.4,.3,.4], [.9,.4,.3,.45]]	[1,2]

(d) AR_2 = CCT (AR_1, first)

fid	oid	[BB]	[FV]	ts
1	1	[10,20,4,5]	[1,2,.7,1.1]	1
1	2	[15,40,7,8]	[.5,.4,.3,.4]	1

Fig. 2. R++ model. (a) A video with 2 frames, (b) R_1 is an R++ table generated from the different attribute types extracted by VCE, (c) AR_1 is an R++ table with extended arrable attributes generated by grouping on oid and ordering on ts, (d) CCT operator applied on AR_1.

are extended arrable attributes, except the grouping attribute oid. Since arrable is backward compatible with the relational model, an R++ relation with extended arrable is also backward compatible with the relational model.

C) Continuous Query Processing for Situation Analysis: The R++ model, extended arrables, and CQL-VA operators (discussed in Sect. 5) are integrated into the continuous query processing component (MavVStream server, an extension of MavEStream [8] of the proposed framework (shown in Fig. 1(d)). The underlying base of MavVStream supports all basic relational operators and aggregates. This is a client-server architecture where clients can submit CQL-VA queries to the MavVStream server. A query plan object is generated from the submitted queries. Each operator in the query plan object is associated with input and output buffers. The system also supports physical and logical windows with flexible window specifications. A multi-threaded feeder is used to feed R++ tuples to the query processor. The complex event processing subsystem has also been integrated to provide primitive event detection capability (based on continuous queries) and compose them further in the future. Once the query results, including the frames where a situation occurs, are generated by the MavVStream Server, these frames can be extracted from the original video and visualized.

5 CQL-VA Operators and the Relational Model

Operators introduced in CQL-VA are based on the requirements discussed in Sect. 2 and can play a role in CQL-VA query optimization in the future. These operators process tuples of an R++ relation like any other relational operator but have some constraints due to their applicability to specific attributes or data types. For example, to filter objects from a video using the relational algebra SELECT (σ) operator, one needs to provide a feature vector or an image from which the feature vector can be extracted. On the other hand, processing a bounding box attribute is relatively simpler using integer values and available wild cards. Joining video streams for object matching (for Q3 in Table 1) also requires dealing with feature vectors. These are not semantically valid for other attributes. This is the same as using operators, such as average, applicable to numeric values. Care has been taken to introduce a minimum number of primitive operators, and composition is used (using closure property) to express larger computations. The CQL-VA operators and conditions, syntax, and complexity are shown in Table 3.

Table 3. CQL-VA Operators and Conditions. Here R$_i$: R++ relation; AR$_i$: R++ relation with extended arrables; gba, aoa: group by and assuming order attributes; N: number of tuples in R_i; M: unique number of objects; th: threshold; S: the complexity of sMatch; a_i: scalar attributes; G: number of rows in AR_i; Nl$_g$, Nr$_g$: average number of tuples in left and right groups in join.

CQL-VA Operators	Syntax	Complexity
Similarity Matching Condition	`sMatch (R`$_1$`.[FV], R`$_2$`.[FV])` `<comparison operator>` th	$\mathcal{O}(S)$
R++ to Arrable	`R2A (R`$_1$`, gba=R`$_1$`.a`$_1$`, aoa=`R_1`.a`$_2$`)`	$\mathcal{O}(N * \log N)$
Compress Consecutive Tuples	`CCT (AR`$_1$`, {`<u>`first`</u>`|last|both})`	$\mathcal{O}(G)$
Consecutive Join	`AR`$_1$ `cJoin` `AR`$_2$ `(condition)`	$\mathcal{O}(G * Nl_g * Nr_g * S)$
CCT Join	`AR`$_1$ `cctJoin` `AR`$_2$ `(condition)`	$\mathcal{O}(G^2 * S)$
Direction	`Direction (AR`$_1$`.[[BB]])`	$\mathcal{O}(M)$

1. Similarity Matching Condition: The similarity matching condition (see the syntax in row 1, Table 3) in CQL-VA compares the similarity of two feature vector attributes of an R++ relation (with or without extended arrables) using the sMatch operator. The sMatch operator computes the similarity score (a numerical distance) between the two feature vector attributes. In the similarity matching condition, the output of sMatch is compared (using comparison operators of the relational model) with a given threshold value (th), which controls to what extent two vectors can be considered similar. Different distance metrics are needed to compute the similarity score since feature vector semantics and size differ based on the VCE algorithm. Two standard distance metrics (values ranging from 0–1), cosine distance (for deep sort) and Euclidean distance (for SIFT and histograms), are supported by CQL-VA. Complexity is shown as $\mathcal{O}(S)$ for this condition in Table 3 as different distance metrics can have different complexity.

2. R++ to Arrable (`R2A`): This operator converts an R++ relation into an R++ relation with extended arrables. The group by (gba) and assuming order (aoa) parameters need to be numeric or categorical attributes. They perform grouping and ordering, respectively, according to AQuery semantics. This operator can apply CCT Join (discussed later in this section) to improve efficiency. It can also be used to perform the running aggregate operations of AQuery on the extended arrables, again to improve efficiency and avoid self-join. The group by operation using hashing takes $\mathcal{O}(N)$ time, and sorting G groups can be approximated by $\mathcal{O}(\log N)$ (upper bound), making the overall complexity $\mathcal{O}(N * \log N)$ for N tuples in an R++ relation or window.

3. Compress Consecutive Tuples (`CCT`): In videos, the same object appears consecutively in multiple frames, and there can be more than one tuple for each object. Computations such as select or join will compare the same object multiple times (quadratically for join). CCT operator is introduced in CQL-VA to reduce the consecutive occurrences of the same object to one or two tuples. This can eventually improve the performance of select and join. CCT takes an R++ relation with extended arrables as input and an option argument, which can be first, last, or both (syntax shown in row 3, Table 3). CCT compresses each extended arrable attribute by keeping the first or last element or the first and last both elements. If the option is first or last, all the extended arrable attributes are converted to their original attribute type (basic/vector). If the option is both, the attribute types remain the same as the input relation. An example of the CCT operator is shown in Fig. 2(d), with the option argument as first. The [fid],oid, and [ts] columns of AR_1 are converted to a numeric type and the columns $[[BB]]$ and $[[FV]]$ are converted to vector type in the relation AR_2. The relational operators such as select (with similarity matching condition and logical operators), join (on scalar attributes and similarity matching condition), and aggregation can be applied to the result set generated by the CCT operator. The computational complexity of CCT is $\mathcal{O}(G)$, where G is the number of tuples in an R++ relation with extended arrables.

4. Consecutive Join (`cJoin`): Although regular join works on extended arrables using the CQL-VA similarity matching condition and existing relational comparison operator, it is still a join that compares each pair from two columns. Situation Q3 in Table 1 can be answered efficiently and accurately without comparing all the pairs of tuples. cJoin is introduced as an alternative to improve accuracy and reduce processing time by performing less number of matches than regular join. cJoin is applied to two R++ relations with extended arrables, and the tuples in each group from both relations are compared until a match is found for a condition. Once a match is found, the rest of the tuples are not compared in that pair of partitions. Although cJoin can be applied to any R++ relations, it provides the best efficiency for feature vector similarity matches from two relations grouped on oid. In general, this will improve efficiency, although in the worst case, similarity matching may succeed only on the last tuple from both sides. Another alternative of cJoin is to drop tuples using CCT and perform a regular join with the similarity matching condition. This may result in incorrect answers. cJoin will provide better accuracy than CCT Join for the same computation, as only 2 tuples can be retained per partition in CCT, which may not match.

(a) AR_1

[fid]	oid	[[FV]]
[1]	1	[**[1,2,.7,1.1]**]
[1,2]	2	[[.5,.4,.3,.4], **[.9,.4,.3,.45]**]

(b) AR_2

[fid]	Oid	[[FV]]
[1,2]	5	[[7,3,.7,1.1], [1,2.2,.9,1.1]]
[2,3,4]	7	[[.3,.1,.7,.6],**[.9,.4,.3,.45],**[.8,.4,.3,.45]]
[4,6]	8	[**[1,2,.7,1.1]**,[1.1,2.2,.7,1.1]]

(c) AR_1 cJoin AR_2

AR_1.oid	AR_2.Oid
1	8
2	7

(d) AR_1 CCT Join AR_2

AR_1.oid	AR_2.Oid
1	8

Fig. 3. (a) AR_1, (b) AR_2 are two R++ tables with extended arrables from two different videos, generated by grouping on oid and ordering on fid, (c) cJoin on AR_1 and AR_2, (d) CCT Join on AR_1 and AR_2. [[BB]] and [ts] columns are not shown due to space constraints. (Color figure online)

An example of the cJoin operation on two R++ relations with extended arrables is shown in Fig. 3(a–c). Here, AR_1 and AR_2 are generated from two different videos. They were grouped on oid and ordered on fid. For simplicity, the [[BB]] and [ts] attributes are not shown here. In AR_1, there are two objects with oid 1 and 2. In AR_2 there are three objects with oid 5, 7, and 8. The result of performing cJoin using the similarity matching condition (see query plan Fig. 4 Q3) with a threshold value of 1 is shown in Fig. 3(c). The elements of [[FV]] column for oid 1 in AR_1 is matched with all the elements of [[FV]] column of AR_2. There is no matching similar feature vectors for oid pairs (1, 5) and (1, 7). However, for the tuples of oid pairs (1, 8) from AR_1 and AR_2, the first element of each feature vector array matches (colored green). In this case, only one comparison is made between the feature vectors in [[FV]] attribute for the rows corresponding to oid 1 and 8. Similarly, for oid 2 and 7 from AR_1 and AR_2, the second element of each feature vector from [[FV]] column was matched. The rest of the elements are not compared in this case. cJoin performs nested loop join instead of hash join as the comparison is not "equality". However, its performance should be much better than a regular nested loop join. The worst case complexity of cJoin is $\sum_{g=1}^{G} \mathcal{O}(Nl_g * Nr_g * S)$, assuming G groups in both (left: l, right: r) relations, each group g with Nl_g and Nr_g number of tuples. S is the complexity of similarity matching condition. This can be simplified to $\mathcal{O}(G * Nl_g * Nr_g * S)$.

5. CCT Join: This operation first applies the CCT operator on an R++ relation with extended arrables and then a regular join (see Fig. 4 Q3 (alternative)). Since CCT reduces the number of elements in each row of the extended arrable attributes to one or two, there are fewer comparisons than cJoin. An example of the CCT Join is shown in Fig. 3(d). Here, only the first and last elements of the arrays in [[FV]] column of AR_1 and AR_2 relation are kept. Hence, for oid 7 the middle element of the feature vector array is dropped and oid 2 and 7 do not come out in the result with threshold 1. The worst case complexity of CCT Join is $\mathcal{O}((G * 2) * (G * 2) * S)$, assuming G groups in both R++ relation, each group $g \in G$ with Nl_g and Nr_g tuples. If first or last is chosen, complexity will be $\mathcal{O}(G * G * S)$. This can be simplified to $\mathcal{O}(G^2 * S)$.

6. Direction (`Direction`): This primitive operator has been introduced to answer queries that require direction (e.g., Q4 in Table 1) and can only be applied to the bounding box attribute of an R++ relation with extended arrables (grouped on object id). Direction outputs one of the 8 directions as an enumerated type in a new column (used like an aggregate function). It is computed by default using the first and last bounding box elements of each extended arrable. It is also possible to compute the direction between the i^{th} and j^{th} element of the extended arrables of bounding boxes, where $i < j$ if specified in the operator explicitly. The complexity of the direction operator is $\mathcal{O}(M)$, where M is the number of unique objects in that particular relation or window.

6 CQL-VA Query Expression for Situation Analysis

Q1 *(Searching for an object using its image)*: To address this situation, the R++ relation generated from an input video stream needs to be searched using the feature vector extracted from the given object image. Select with similarity matching condition in the where clause can answer this situation (see query plan in Fig. 4).

Q2 *(Busy period in a public area):* This situation requires counting the number of unique objects (e.g., person/cars) in a premise every 'n' seconds. An R++ relation (within the 'n' second window or whole video), can be converted

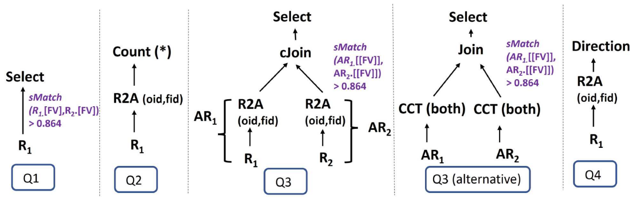


Fig. 4. Query plan for Table 1 situations. Two plans are shown for Q3. AR$_i$ is generated after applying `R2A` on R++ table R$_i$ as shown in Q3. Only similarity matching conditions are shown.

to an R++ relation with extended arrables by grouping on oid and ordering on fid using the R2A operator. Then the number of rows in the resultant relation can be counted. A busy period can be identified by projecting the timestamps with the maximum number of unique objects in the relation. The query expression is shown below (see query plan in Fig. 4).

Q3 *(Presence of the same person in two/more videos):* The R++ relation generated from two different videos must be joined using a similarity matching condition to address Q3. A query expression using cJoin for Q3 is shown below. An alternative query plan using CCT Join is shown in Q3 (alternative) in Fig. 4. Q3 can also be expressed using a regular join with the similarity matching condition.

Q2 Expression:

```
Select count(*)
From
(R2A(R1, R1.oid, R1.fid)) AR1
Where R1.label = "person"
```

Q3 Expression:

```
Select AR1.oid, AR2.oid
From
(R2A(R1, R1.oid, R1.fid)) AR1
cJoin
(R2A(R2, R2.oid, R2.fid)) AR2
on sMatch (AR1.[FV], AR2.[FV]) > .864
```

Q4 *(Which direction an object moved?):* Query formulation is similar to Q2, where `Direction` operator (instead of the count operator) can be used in the Select clause (see query plan in Fig. 4).

7 Experimental Results

Dataset and Experimental Setup: In this work, three different datasets: CamNeT [19], MavVid [1], and MavVidR [2] have been used for experiments. The CamNeT dataset contains video footage of people passing by a place (e.g., hallway) in a complex environment (e.g., low light, shadow). The MavVid dataset was prepared in ITLAB by capturing entry and exit videos with situations Q1-Q4 from Table 1. Videos 43, 37, and 39 from the MavVid dataset were concatenated with arbitrary videos from the CamNeT dataset and videos (of the Capitol riot incident and a soccer match) containing arbitrary situations to generate videos 43^{++}, 37^{++}, and 39^{++} in the MavVidR dataset. The dataset description in terms of video length and number of tuples generated by the R++ relation is shown in Table 4. All the videos in this table were generated at a 30-frame-per-second rate. Although two videos can be of different lengths, the number of tuples varies significantly depending upon the objects present in a video (e.g., Video 57 has fewer tuples than 41 even though their length is the same). This is important as the query processing time depends on the number of tuples processed.

The MavVStream prototype was implemented in JAVA. Videos were pre-processed using YOLO and deep-sort implementation in Python on an NVIDIA Quadro RTX 5000 GPU with 10 GB memory. Query processing experiments were performed on the same machine with 2 Intel Xeon processors (48 cores, 748 GB main memory).

Table 4. Dataset description

Dataset	V_{id}	Length	Tuples
MavVid [1]	39	41 s	173
	37	49 s	6442
	41	28 s	2447
	43	37 s	382
	53	50 s	2218
	57	28 s	1645
MavVidR [2]	37^{++}	82 s	7520
	39^{++}	56 s	1370
	43^{++}	56 s	1857
CamNeT [19]	1	24 min	11691
	23	20.5 min	15700
	29	22 min	11035

Result Comparison: None of the existing systems can completely address all the situations from Table 1. Even though some systems can partially address Q2 and Q4 (as shown in Table 2), these systems and their datasets are not open-source, making comparison difficult. No baseline system can answer Q3 to the best of our knowledge. Besides, the proposed approach differs from the existing systems (e.g., query processing vs. machine learning, relational vs. graph model). Comparing them will not appropriately highlight the differences between the systems.

CQL-VA Query Results and Evaluation: CQL-VA query evaluation using accuracy, robustness, efficiency, and scalability are discussed below.

A) Accuracy: Accuracy is computed by manually generating the ground truth. There can be two types of ground truth: Ground Truth (GT) from the actual video and Ground Truth from the pre-processed video (GT(vce)). There is a significant difference between GT and GT(vce), as the information (in terms of situations present) in the actual video differs from the pre-processed video. For example, the same oid may be given to two objects (having the same colored clothes, shape, etc.), or the same object can have two different oids in two consecutive frames (because of being recognized as two different objects). The vision community is investigating these challenges, which differ from the query processing problems discussed here. Hence, this paper computes the query accuracy using GT(vce) (termed as Acc(vce)).

Q1 *(Searching for an object):* Q1 experiments were conducted on MavVid Dataset (see Table 5, rows 1–3) using a time-based disjoint window of 40 s. Since Q1 requires the similarity matching condition to be used with select and is a tuple-by-tuple comparison, different window sizes do not have any effect, and the window size was set to take the whole video as a window for most of the videos in MavVid (except 53). Feature vectors of object images were extracted separately using the same VCE algorithm. The videos were searched with different *th* as a small deviation in *th* affects accuracy significantly (feature vectors vary significantly in the video in consecutive frames). The accuracies shown in Table 5 are obtained based on best *th* (set experimentally) for which the highest accuracy was obtained. The best *th* for videos 41, 43, and 53 are .85, .864, and .835 respectively. Changing the *th* value of video 41 from .85 to .7990 drops the accuracy to 0%. This is true for other videos and represents the sensitivity of the *th* value (automated *th* determination is an open problem).

Q2 *(Aggregation)* and **Q4** *(Direction):* Q2 and Q4 experiments were performed on the CamNeT and MavVid datasets using the query formulation shown in Sect. 6 and the query plan in Fig. 4. The results are shown in Table 5. A time-based disjoint window 40 s was used for the videos of the MavVid dataset. These experiments evaluated if the newly introduced CQL-VA operators (R2A, Direction) can perform the computations correctly. Hence, for smaller videos of the MavVid dataset, the window was set so that the whole video was a window (except video 53). For Q2 and Q4, accuracy is 100% for videos 41, 43, and 53. A bigger dataset, CamNeT, with a larger window size (10 min.), was also used for evaluation, and accuracy was 100% for Q2 and Q4. These experiments show that the CQL-VA operator computations are correct.

Table 5. Q1, Q2, and Q4 Accuracy

V_{id}	Query	Acc(vce)
41	Q1	100%
	Q2	100%
	Q4	100%
43	Q1	100%
	Q2	100%
	Q4	100%
53	Q1	100%
	Q2	100%
	Q4	100%
1	Q2	100%
	Q4	100%
23	Q2	100%
	Q4	100%
29	Q2	100%
	Q4	100%

Q3 *(Presence of same objects in two videos):* As mentioned earlier, Q3 is important for analyzing and monitoring the entry and exit of individuals in a premise. To validate that, entry and exit videos from the MavVid datasets were used to determine the presence of the same persons in two or more videos. A sample result of Q3 (using proposed cJoin) is shown in Fig. 5. Here, in video 43, people enter through a corridor; in video 39, people exit through a door, and both have one common person. In Fig. 5(c), using cJoin, the same person's presence at different times in 39 and 43 was identified. The experiments for Q3 were conducted using the alternative query plans shown in Fig. 4 along with the regular join (with similarity matching condition).

As no existing system addresses Q3, we have considered regular join with the similarity matching condition as our baseline and compared the accuracy and performance of Q3 using cJoin and CCT Join. The th value was the same for the similarity matching condition in join for all the joins. Time-based disjoint windows of 10 s, 20 s, and 40 s were used. The window sizes were multiplied by two to evaluate the effect on accuracy with varying window sizes. In Table 6, experiment results for only 20 s, and 40 s windows are shown due to space constraints. The number of objects in each video and common object pairs from the two videos used for joining (true positives) are also shown. Performance improvement for cJoin and CCT Join are shown with respect to regular join. A self-join was performed on video 37 to verify the correctness of the different join operators, and accuracy was 100% for all the joins (row 1, Table 6). For

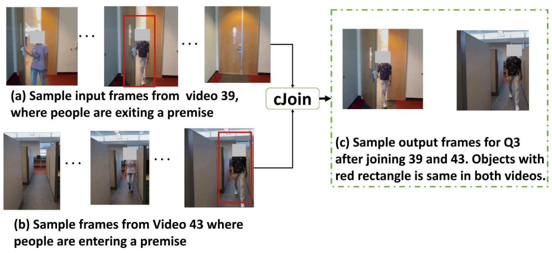

(a) Sample input frames from video 39, where people are exiting a premise
cJoin
(b) Sample frames from Video 43 where people are entering a premise
(c) Sample output frames for Q3 after joining 39 and 43. Objects with red rectangle is same in both videos.

Fig. 5. Sample results of Q3 for 43 $\bowtie$ 39.

a 20 s window (in Table 6 rows 2–8) cJoin obtained the highest accuracies for 6 video pairs. In these videos, regular join brings out all the object pairs having feature vector similarity above the *th*. Hence, there are more false positives as the same object is considered similar to multiple objects because of feature vector sensitivity. Since cJoin stops matching the pairs of objects once the similarity matching condition is satisfied, fewer false positives are produced. On the other hand, CCT Join has the second-highest accuracy for four pairs of videos. However, it has the least query processing time for all the video pairs except for 39 and 43 (row 5 in Table 6). This happens because these videos are the smallest (regarding the number of tuples), and an overhead for grouping, ordering, and dropping tuples for CCT is introduced.

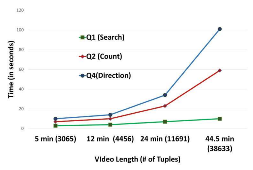
Table 6. Q3 accuracy comparison. V_{id_i}: Video id, C_O: # of common object pairs from two videos, W_s: window size.

W_s	$V_{id_1} \bowtie V_{id_2}$	# of Objects		C_O	Acc(vce)			Performance improvement	
		V_{id_1}	V_{id_2}		Join	cJoin	CCT Join	cJoin	CCT Join
20 s	$37 \bowtie 37$	5	5	5	100%	100%	100%	40%	**75%**
	$37 \bowtie 39$	5	3	1	67%	**87%**	73%	66%	**70%**
	$37 \bowtie 43$	5	3	1	**87%**	**87%**	60%	67%	**70%**
	$37 \bowtie 57$	5	3	1	20%	**73%**	67%	89%	**86%**
	$39 \bowtie 43$	3	3	2	**89%**	78%	**89%**	**33%**	-57%
	$39^{++} \bowtie 43^{++}$	14	23	2	91%	**99%**	98%	78%	**96%**
	$37^{++} \bowtie 39^{++}$	6	14	2	85%	**92%**	90%	74%	**83%**
	$37^{++} \bowtie 43^{++}$	6	23	2	93%	**95%**	83%	66%	**83%**
40 s	$37 \bowtie 39$	5	3	1	67%	**93%**	80%	14%	**19%**
	$37 \bowtie 43$	5	3	1	**87%**	67%	67%	29%	**37%**
	$37 \bowtie 57$	5	3	1	20%	73%	**80%**	**86%**	84%
	$39 \bowtie 43$	3	3	2	89%	**100%**	**100%**	**50%**	-40%
	$39^{++} \bowtie 43^{++}$	14	23	2	93%	**96%**	94%	82%	**85%**
	$37^{++} \bowtie 39^{++}$	6	14	2	82%	**94%**	92%	48%	**71%**
	$37^{++} \bowtie 43^{++}$	6	23	2	93%	**96%**	89%	29%	**73%**

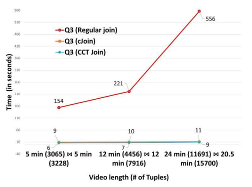
When the window size was increased twice (40 s) (shown in Table 6, rows 9–15), cJoin obtained the highest accuracies for five pairs of videos. CCT Join accuracy was second highest among those video pairs. For video pairs 37 and 57, cJoin and CCT obtain 73% and 80% accuracy, respectively, whereas regular join obtains 20% accuracy. This happened because cJoin and CCT Join make fewer comparisons, generating fewer false positives. For a 40 s window, CCT Join reduces the query processing time for four pairs of videos and cJoin for two pairs of videos. cJoin improves accuracy and reduces query processing time in the majority of cases. CCT Join reduces the query processing time of cJoin further by sacrificing the highest 13% accuracy (still, the accuracy is 80%). The trend is similar in the 10 s window as well. The average accuracy for Join, cJoin, and CCT Join was 84%, 89%, and 89%, respectively, for the 10 s window. cJoin, and CCT Join improved the join performance on average by 93% and 99%, respectively, for a 10 s window. The above experiments showcase that cJoin is the ideal operator for both improving accuracy and reducing performance time. CCT Join may sacrifice some accuracy for a better performance (accuracy vs performance tradeoff). However, for real-time processing of large videos, this operator might be ideal. Even the average accuracy of cJoin and CCT Join is 85% and 80%, and the average performance improvement is 47% and 50% for all three window sizes, respectively. Given the inconsistency of feature vectors, this accuracy and performance improvement is good.

B) Robustness: A query formulation is highly robust if the same accuracy is obtained *with and without the presence of other situations* in the video. The MaVidR dataset was used to validate the robustness of our queries using 10 s, 20 s, and 40 s windows for Q3, as it is the most complex query. The experiment results are shown in Table 6 (rows 6–8 and 13–15). Here, the true negatives increased, but true positives remained the same because of additional objects in the concatenated video part. For the video pairs $(39^{++}, 43^{++})$, $(37^{++}, 39^{++})$, and $(37^{++}, 43^{++})$, in both 20 s and 40 s window in Table 6, cJoin and CCT Join improves accuracy from Join as they do not bring out the additional object pairs as false positives. They also improve performance by at least 29% and 71% for these video pairs. This shows that the CQL-VA operators can identify a situation correctly in the presence of arbitrary situations and optimize performance.

C) Scalability: Scalability measures query performance when the video size increases by a fixed length. Different length videos (maximum 44.5 min.) were used for scalability experiments. Since there is a scarcity of large videos containing all the situations of interest, Videos 1 and 23 of the CamNeT dataset were trimmed and increased by a fixed length. 5 and 12 min videos were created from Videos 1 and 23 by taking the first 5 and 12 min from each video, and a 44.5-min video by merging videos 1 and 23. The purpose of this was to increase the size of the videos 2 times to understand how the query processing time increases. In Fig. 6(a), Q1, Q2, and Q4 performance is shown. With increased video size, the query processing time increases linearly for Q1, Q2, and Q4. The curve values vary based on the operator used and the computation time. Since Q1 has only select, it takes the least time among all the queries. Q3 performance was measured in Fig. 6(b) by joining the same-size videos sampled from 1 and 23 and joining 1 and 23. As mentioned earlier, the regular join (our baseline) time (maximum 556 s and 98%) is significantly higher than cJoin and CCT join as the number of comparisons made is higher. This validates our intuition for introducing these operators. **New CQL-VA operators introduced perform more than an order of magnitude faster than the regular Join.**



(a) Q1, Q2, Q4



(b) Q3 with different joins

Fig. 6. Query efficiency and scalability on CamNeT dataset using 10 min. time-based disjoint window

8 Conclusion

In this paper, the components for automated video analysis were identified, and a framework was proposed. This paper shows how a few new primitive operators can detect diverse situations. This paper also demonstrates that query-based situation monitoring can be achieved using established and low-risk approaches. A complete generalized situation analysis system supporting more primitive operators and more complex situation analysis is under development.

Acknowledgements. This work was supported by NSF award #1955798 and #1916084. The authors would like to thank Dr. Abhishek Santra for his constructive suggestions.

References

1. MavVid Dataset (2023). itlab.uta.edu/downloads/mavVid-datasets/MavVid_v2.zip
2. MavVidR Dataset (2023). itlab.uta.edu/downloads/mavVid-datasets/MavVidR_v2.zip
3. Aved, A.J., Hua, K.A.: An informatics-based approach to object tracking for distributed live video computing. Multimedia Tools Appl. **68**(1), 111–133 (2014)
4. Bastani, F., et al.: MIRIS: fast object track queries in video. In: ACM SIGMOD, pp. 1907–1921 (2020)
5. Baumann, P., Misev, D., Merticariu, V., Huu, B.P.: Array databases: concepts, standards, implementations. J. Big Data **8**(1), 1–61 (2021)
6. Chao, D., Koudas, N., Xarchakos, I.: SVQ++: querying for object interactions in video streams. In: 2020 ACM SIGMOD, pp. 2769–2772 (2020)
7. Dönderler, M.E., Şaykol, E., Arslan, U., Ulusoy, Ö., Güdükbay, U.: BilVideo: design and implementation of a video database management system. Multimedia Tools Appl. **27**(1), 79–104 (2005)
8. Jiang, Q., Adaikkalavan, R., Chakravarthy, S.: MavEStream: synergistic integration of stream and event processing. In: ICDT, p. 29 (2007)
9. Kang, D., Bailis, P., Zaharia, M.: BlazeIt: optimizing declarative aggregation and limit queries for neural network-based video analytics. VLDB **13**(4), 533–546 (2019)
10. Kang, D., Emmons, J., Abuzaid, F., Bailis, P., Zaharia, M.: NoScope: optimizing deep CNN-based queries over video streams at scale. PVLDB **10**(11), 1586–1597 (2017)
11. Lerner, A., Shasha, D.: AQuery: Query language for ordered data, optimization techniques, and experiments. In: PVLDB, pp. 345–356 (2003)
12. Lu, C., Liu, M., Wu, Z.: SVQL: a SQL extended query language for video databases. Int. J. Database Theory Appl. **8**(3), 235–248 (2015)
13. Mathew, S., Varia, J.: Overview of Amazon web services. Amazon Whitepapers **105**, 1–22 (2014)
14. Redmon, J., Farhadi, A.: YOLOv3: an incremental improvement. arXiv preprint arXiv:1804.02767 (2018)
15. Wojke, N., Bewley, A., Paulus, D.: Simple online and realtime tracking with a deep association metric. In: IEEE ICIP, pp. 3645–3649 (2017)
16. Xiong, P., Zhan, H., Wang, X., Sinha, B., Wu, Y.: Visual query answering by entity-attribute graph matching and reasoning. In: CVPR, pp. 8357–8366 (2019)
17. Yadav, P., Curry, E.: VidCEP: complex event processing framework to detect spatiotemporal patterns in video streams. In: IEEE BigData, pp. 2513–2522. IEEE (2019)

18. Zhang, E., Daum, M., He, D., Haynes, B., Krishna, R., Balazinska, M.: EQUI-VOCAL: synthesizing queries for compositional video events from limited user interactions. VLDB **16**(11), 2714–2727 (2023)
19. Zhang, S., Staudt, E., Faltemier, T., Roy-Chowdhury, A.K.: A camera network tracking (CamNeT) dataset and performance baseline. In: IEEE WACV, pp. 365–372 (2015)