DeepVQL: Deep Video Queries on PostgreSQL

Dong June Lew
Kunsan National University
rdj2010075@kunsan.ac.kr

Kihyun Yoo
Kunsan National University
khyoo1221@gmail.com

Kwang Woo Nam*
Kunsan National University
kwnam@kunsan.ac.kr

ABSTRACT

The recent development of mobile and camera devices has led to the generation, sharing, and usage of massive amounts of video data. As a result, deep learning technology has gained attention as an alternative for video recognition and situation judgment. Recently, new systems supporting SQL-like declarative query languages have emerged, focusing on developing their own systems to support new queries combined with deep learning that are not supported by existing systems. The proposed DeepVQL system in this paper is implemented by expanding the PostgreSQL system. DeepVQL supports video database functions and provides various user-defined functions for object detection, object tracking, and video analytics queries. The advantage of this system is its ability to utilize queries with specific spatial regions or temporal durations as conditions for analyzing moving objects in traffic videos.

PVLDB Reference Format:
Dong June Lew, Kihyun Yoo, and Kwang Woo Nam. DeepVQL: Deep Video Queries on PostgreSQL . PVLDB, 16(12): 3910 - 3913, 2023.
doi:10.14778/3611540.3611583

1 INTRODUCTION

With the recent advancements in mobile and camera devices, a tremendous amount of videos are being generated, shared, and used at an incredible speed. From the perspective of applications, more cameras are being deployed for traffic monitoring or surveillance systems than ever before. The speed at which events in a video is recognized and processed is exceeding the limits of human capabilities. As a realistic alternative for recognizing and making judgments about event situations in videos, researches on deep learning technology are gaining popularity[5].

Traditional systems for managing and supporting deep learning-based situational judgment for large amounts of videos were loosely-coupled systems of independent systems. For example, a system that processes and stores large video streams and performs deep learning-based object recognition and tracking using vision processing, and then sequentially uses this information to recognize specific situations and detect or retrieve for event frames. However, the practical demands of dealing with massive video streams or numerous video files have changed with the times, necessitating a change in the direction of research on existing video processing systems. [9] In recent years, research on new systems that support declarative query languages similar to SQL has emerged[1, 7, 9, 14, 15].

The advantage of declarative query languages for video processing is that users can present their requirements in an integrated manner through easy logical queries, and the system can incorporate various query optimization ideas commonly used in traditional DBMS technologies to improve video data processing performance. Lu et al.[9] showed that the query performance can be improved by adjusting the order of probabilistic filters in the query plan with different selectivity. Since their significant paper, various systems such as BlazeIt[6], SVQ[7, 14], MIRIS [1], and EVA[15] have been actively proposed. These systems have successfully presented on proofing their novel techniques for exploratory video analytics. However, they have only focused on implementing their ideas by developing their own systems since the existing systems do not support new deep-learning-combined operators, but also does not video database management systems. The proposed DeepVQL system have been developed on the PostgreSQL system, an open-source DBMS that is actively used and has powerful DBMS features. We propose various novel user-defined functions(UDFs) to support video database functionality on PostgreSQL, while supporting deep learning detection and video analytics queries.

One of most useful applications of declarative query languages is a traffic video monitoring system. Traffic videos contain very dynamic information about the major infrastructure of the city's roads, while also requiring highly analytical queries for specific spatial regions or temporal period, such as accident detection or aggregation. Assume a deep content-based query for traffic videos that calculates how many vehicles pass through a specific spatial area such as crosswalks even within a specified time. The following is an example query used for our DeepVQL system.

```
SELECT count(*)
FROM D_MProcess('trafficvideo','vehDetector') t, crosswalk c
WHERE t.className = 'car' AND D_Cross(t.itraj, c.geo)
```

DeepVQL users can query this UDF-extended SQL to detect and track vehicles through pre-trained deep learning models, and then retrieve and aggregate the number of vehicles passing through crosswalk areas of the video screen. This query shows two major advantages of DeepVQL. Firstly, user can easily make a declarative query using the *D_MProcess* UDF to detect and track vehicles on a traffic video using a pre-defined deep learning model called 'vehDetector'. If the user has pre-registered a MaskRCNN model trained using MS COCO[8] data as 'cocoDetector', they can easily apply different models to the video with a simple query change such as $D\_MProcess('trafficvideo','cocoDetector')$. The second advantage is that DeepVQL supports spatial join queries with existing legacy data already stored in PostgreSQL. In the above example query, the geometry information of the existing crosswalks in a city is stored in the "geo" column of the "crosswalk" table using PostGIS[10], and the trajectories of the detected vehicle's image coordinates are stored in the "itraj" column, and they can be joined in general SQL syntax. DeepVQL supports spatial query processing

*Kwang Woo Nam is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611583

in *D_Cross* UDF by calling PostGIS UDFs, which supports spatial extensions for PostgreSQL. In traditional video analytics systems, handling such queries would likely require the implementation of very complex and high-level new UDF features.

In section 2, we discuss the system architecture of DeepVQL demonstration system and deep moving objects databases. In next section, We will describes deep video queries which are supported in DeepVQL system such as spatial video queries, spatio-temporal video queries, and spatial traveling video queries. In section 4, we show demonstration scenarios with real traffic videos.

2 DeepVQL SYSTEM

DeepVQL is a SQL-based querying system that supports deep video queries for videos on PostgreSQL. In this section, we will describe the system architecture and how to support various deep objects storage and video queries on DeepVQL system. DeepVQL demonstration system has five components as shown in Figure 1. It consists of DeepVQL on PostgreSQL, DeepVQL Detection Task Manager, DeepVQL Video Repository, DeepVQL Web API Services, and Web Demonstration UIs.

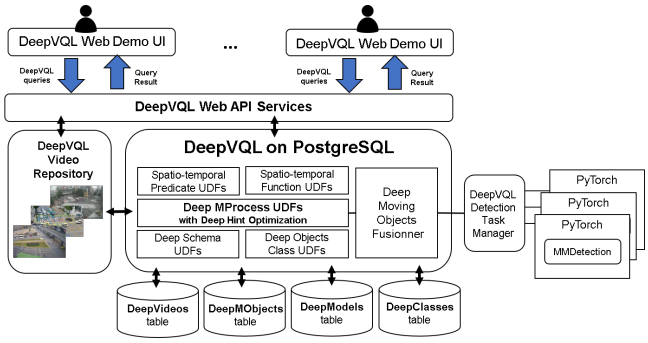

Figure 1: System Architecture for DeepVQL Demonstrations

DeepVQL on PostgreSQL is a main part of DeepVQL system that includes Deep MProcess UDFs, Spatio-temporal Predicate and Function UDFs, Deep Schema UDFs, Deep Objects Class UDFs, and Deep Moving Objects Fusionner. Deep MProcess UDFs is a set of deep moving objects processing UDFs with a gateway function *D_MProcess*. Spatio-temporal Predicate and Function UDFs are sets of querying deep moving objects in the videos, and Deep Schema UDFs and Deep Objects Class UDFs are sets of functions for inserting and managing the objects and metadata in DeepVideos, DeepModels, DeepClasses and DeepMObjects tables.

DeepVideos The DeepVideos table is a metadata set of videos that are stored and managed in DeepVQL Video Repository. A video in DeepVideos can be identified by a unique video identifier *videoid*, a video resource location identifier *videoUri* for the repository access, or a short unique video name *videoName* for DeepVQL query users.

DeepModels The DeepModels table is a metadata set of pre-trained object detection and tracking models that are supported by Fusionner, Deep Detection Task manager and PyTorch/MMDetection[3]. DeepVQL can support MMDetection-powered 26+ detection models and 100+ retrained models for deep video queries. The query users simply apply the pre-trained model for videos using a unique model identifier *modelid* or unique model name *modelName* in DeepModels table. After distributed object detection by task manager, Deep Moving Objects Fusionner perform the tracking and merging job for the detected objects into moving objects using a self-implemented and efficient algorithm.

DeepClasses The DeepClasses table is a set of class category information tuple (*cid*, *className*, *modelid*, *modelName*) about all possible classes which can be detected by the pre-trained models. Suppose that a pre-trained RCNN model trained by Microsoft COCO dataset. When a moving object *m* was detected and tracked by the RCNN and COCO dataset trained model, *m* can be classified into one of 80 COCO classes including 'person', 'car', and 'bus'. In DeepClasses, a class can be identified by a unique class identifier *cid* or a unique class name *className* with a model identifier *modelid* or *modelName* for query users.

DeepMObjects The DeepMObjects table M is a set of moving objects which are detected moving objects in videos by pre-trained deep learning models. A moving object m in M is a tuple (oid, cid, $className$, $modelid$, $t\_modelid$, $videoid$, $itraj$, $frameids$, $timestamps$). Each moving object in M is uniquely identified by an object ID (oid). A moving object m has a class ID (cid) and a class name ($className$), which are detected and merged into moving objects by the pre-trained model ($modelid$) and the tracking model ($t\_modelid$) from the video ($videoid$). The detected moving object can change the coordinates for every frame. $itraj$ is a sequence of 2D coordinates for storing representative location in the frame image of the moving object. The default way is that the representative coordinates correspond to the center point of the objects' bounding boxes. Also, we need to manage in which frame and time the objects appeared, moved, and disappeared. $frameids$ is a sequence of frame numbers which the detected object exist. To efficiently retrieve moving objects for temporal predicate queries, DeepMObjects table need to store temporal information on the video in $timestamps$.

Suppose that a video showing traffic condition at the intersection in Warsaw is newly inserted, and that the task of counting the number of vehicles entering and leaving from a certain direction is queried. The following shows an example of downloading a video from a website and inserting it into the DeepVQL system.

```
SELECT D_Video_Append('http://example.edu/warsaw.mp4',
        'id://warsaw.mp4', 'yolo_coco', 'ByteTracker');
```

After verifying that the video and the deep models are valid in DeepVideos and DeepModels metadata table, DeepVQL system would detect all objects in the video by DeepVQL Detection Task Manager using PyTorch/MMDetection, and then the detected frame objects are merged as moving objects on the video sequence by Deep Moving Object Fusionner using in-house implemented tracking algorithms. Deep Moving Object Fusionner currently supports a ByteTracker[16] by default algorithm, and additionally support DeepSORT[13] and OCSORT[2] algorithms for objects tracking. After merging step, Deep *D_MProcess* UDF stores all moving objects into the DeepMObjects table, which is an integrated fundamental table for deep video queries. Every moving object in DeepMObjects has information about which pre-trained model and tracking algorithm were used in *modelid* and *t_modelid*, and this information

can be used for queries. DeepVQL user may give a query with a temporal range predicate and a specific deep learning model or class.

DeepVQL is distinguished from previous work in that developed on moving objects data and query model for storing and querying the detected moving objects. In general approach, all detected objects are stored in separate rows. Then, a spatial predicate operation is carried out on each moving object to determine if their trajectory data has passed the crosswalk to count the number of vehicles that have crossed the crosswalk. However, our system stores the trajectory data for each moving object in one row, and can process the spatio-temporal predicates efficiently.

3 DEEP VIDEO QUERIES

Our DeepVQL system supports three types of video queries. Each video query performs a spatio-temporal relation operation on the DeepMObjects table M and returns a moving object set M' satisfying.

Spatial Video Queries Spatial Video Query enables extraction of semantic information through spatial relationship queries between moving objects and spatial regions expressed in Well-Known Text (WKT) within a traffic video. For example, to obtain information about a vehicle passing a certain lane in a traffic video, it would be sufficient to investigate the trajectory data of the vehicle and the spatial relationship between the vehicle and the lane. To extract semantic information, we have implemented operations to establish relationships between moving object m and spatial regions within a traffic video. The query is expressed as follows.

```
SELECT oid
FROM D_MProcess('id://warsaw.mp4','yolo_coco','DeepSORT')
WHERE D_Cross(itraj, 'LINESTRING (532 367, 568 493)')
      AND className = 'car'
```

The spatial relationship operation *D_Cross* is performed on the records of DeepMObjects M. The *D_Cross* function takes as input the pixel trajectory data of a moving object, referred to as "itraj", and a line (WKT), and returns a boolean value indicating whether the given WKT and itraj intersect. Through this, we can obtain a set of moving objects M' that satisfy the spatial predicate Cross. As a result of the query, we can obtain *oid*s for vehicles passing a particular lane in a traffic video.

Spatio-temporal Video Queries Spatio-temporal video queries return moving objects that satisfy both spatial and temporal predicates. Such queries encompass both space and time criteria and are frequently used for monitoring traffic videos. For example, people monitoring traffic videos may not need to watch the entire video all the time. They might only need to monitor specific time ranges or locations to observe specific events. Let's reconsider an example query for obtaining information about vehicles passing through a specific lane. Traffic video monitoring observers may wish to check only the vehicles passing through a particular lane for a specific period. For this purpose, we provide a temporal query. The query is as follows.

```
SELECT oid
FROM D_MProcess('id://warsaw.mp4','yolo_coco','OCSORT')
WHERE D_Cross(itraj, 'LINESTRING (827 353, 918 480)')
     AND D_TOverlap(timestamps,$1,$2) AND className = 'car'
```

In this query, an additional *D_TOverlap* function is performed on the records of DeepMObjects M. This function takes the timestamps of the moving object and the queried time range ($1,$2) as inputs. It returns a boolean value to indicate whether there is any overlapping time between the timestamps and the values of $1,$2. By utilizing this query, we can obtain vehicle IDs for those that passed through a specific lane during the given time range.

Spatial Traveling Video Queries In the following query, we consider a scenario where multiple spatial regions are used as conditions to aggregate information about vehicles moving in a specific direction. Specifically, we aim to gather data on the number of vehicles that are moving in a particular direction within the specified regions. The query is as follows.

```
SELECT count(*)
FROM D_MProcess('id://warsaw.mp4','yolo_coco','ByteTracker')
WHERE D_Travel(itraj, ARRAY['LINESTRING (532 367, 568 493)',
     'LINESTRING (827 353, 918 480)']) AND className = 'car'
```

After executing *D_MProcess*, the *D_Travel* function verifies the sequential passage of the lines provided in the extracted moving object records. This allows for the aggregation of vehicle driving direction. Additionally, by reversing the spatial constraint lines in this query, it becomes possible to detect anomalies. Vehicles that do not pass through the lines in the correct sequence can be identified as those driving in the opposite direction, indicating the presence of wrong-way vehicles.

The next aspect to be considered is when the object detection and tracking for traffic videos will be performed in the database. This can be divided into three main perspectives:

(1) **On the appending phase**: This involves pre-processing the entire video beforehand when inserting video information into DeepVQL via the fuction *D_video_append*. Since the processing is completed beforehand, query speed is increased. However, it is not possible to pre-process all detection and tracking algorithms, and only the designated ones are pre-processed.
(2) **On the querying phase**: For videos that are not previously registered, the *D_MProcess* function processes the entire video at once if no options are specified during its initial execution. While the first query may take longer, subsequent queries are faster. This approach provides the advantage of being able to actively adapt to a variety of detection and tracking algorithms.
(3) **On the querying phase with spatio-temporal range**: When there are temporal predicates in the query, only the part of the video that includes the give period and margin parts are processed, and the query results are returned quickly. This approach also provides the advantage of being able to actively adapt to a variety of detection and tracking algorithms, but requires complex algorithms and is currently under development.

This paper mainly describes on the querying phase. During execution, the *D_MProcess* function checks whether data satisfying the query for the specified video is already stored. If available, it returns the moving object records; otherwise, it pre-processes the target video, stores the extracted moving objects in DeepMObjects, and then returns the records. If there are spatio-temporal predicate functions, it performs the functions on the returned moving objects records.

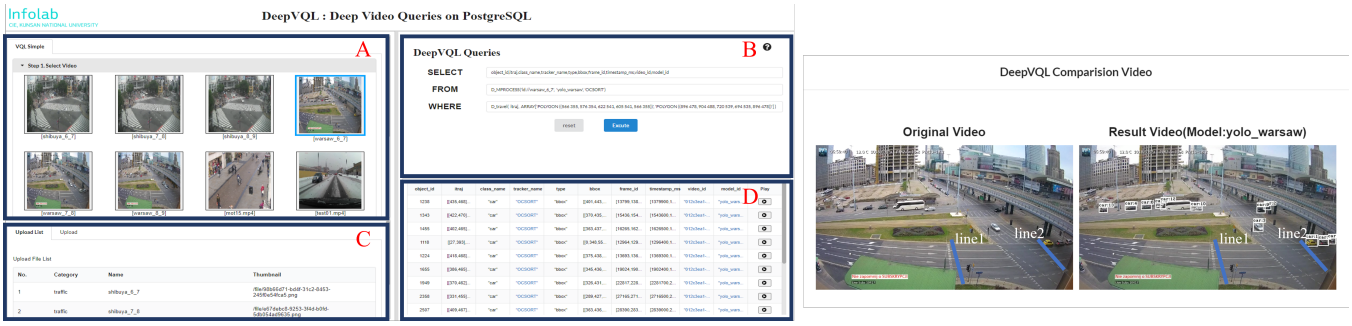

Figure 2: DeepVQL demonstration using traffic videos

4 DEMONSTRATION

This section describes how to interact with the DeepVQL system to query traffic videos that include the previously introduced queries.

Web DeepVQL The user interface of the system is depicted in Figure 2. It consists of four main components: the query selection window (A), the query window (B), the video upload window (C), and the result window (D). The query selection window (A) provides settings for spatial queries, spatio-temporal queries, and spatial traveling queries. Each step provides a list of options for the user to choose from, including a list of videos, model lists, tracking algorithm lists, spatial predicate operation lists, and column lists. Each of the sample videos in the video list uses video from MIRIS [1]. The videos provided by MIRIS[1] are traffic videos from Shibuya in Japan and Warsaw in Poland, each with a duration of approximately three hours. The object detection models, including faster-RCNN[12], YOLO[11], and Mask-RCNN[4], were trained using the training datasets provided by MS COCO[8] and MIRIS[1]. The tracking algorithms provided by our system are ByteTracker, OCSORT, and DeepSORT[2, 13, 16].

When the user selects elements from these lists, they are reflected in the query window (B). The query window (B) automatically generates scripts in the SELECT, FROM, and WHERE clauses based on the selections made in the query selection window (A). The query window (B) contains a "?" icon with examples that users can refer to when querying. These examples include the three types of query examples described earlier, allowing users to execute the query clauses in query window B as they are or make slight modifications to test their queries. The user can reset the selected steps using the "reset" button. By clicking the "execute" button, the contents of each SQL clause are transmitted to the DeepVQL. The DeepVQL executes the query and displays the results in the results window (D). The video upload window (C) is divided into two tabs: the "upload list" tab and the "upload" tab. In the "upload" tab, users can select a video file they possess by clicking the "choose file" button, and then submit the video to the server by clicking the "submit" button. Once the video is uploaded, it will be reflected in the video list, and the video's thumbnail will be created, allowing the user to select it in step 1 of the query selection window (A). The query results are displayed in the table in the results window(D), and clicking the play button on the right allows the video player to pop up and view clip images of moving objects. (See http://deepvql.urbanai.net)

ACKNOWLEDGMENTS

This work was supported by the KAIA grant (Grant RS-2022-00143336), ETRI grant (21ZR1200), and the NRF grant (No.2020R1F1A1048432) funded by the Korea government (MOLIT, MSIT, DNA-based national intelligence core technology development).

REFERENCES

[1] Favyen Bastani, Songtao He, Arjun Balasingam, Karthik Gopalakrishnan, Mohammad Alizadeh, Hari Balakrishnan, Michael Cafarella, Tim Kraska, and Sam Madden. 2020. Miris: Fast object track queries in video. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1907–1921.

[2] Jinkun Cao, Jiangmiao Pang, Xinshuo Weng, Rawal Khirodkar, and Kris Kitani. 2023. Observation-centric sort: Rethinking sort for robust multi-object tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9686–9696.

[3] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, et al. 2019. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155* (2019).

[4] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*. 2961–2969.

[5] Daniel Kang, Peter Bailis, and Matei Zaharia. 2018. Blazeit: Optimizing declarative aggregation and limit queries for neural network-based video analytics. *arXiv preprint arXiv:1805.01046* (2018).

[6] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. Challenges and Opportunities in DNN-Based Video Analytics: A Demonstration of the BlazeIt Video Query Engine.. In *CIDR*.

[7] Nick Koudas, Raymond Li, and Ioannis Xarchakos. 2020. Video monitoring queries. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[8] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*. Springer, 740–755.

[9] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. 2018. Accelerating machine learning inference with probabilistic predicates. In *Proceedings of the 2018 International Conference on Management of Data*. 1493–1508.

[10] Paul Ramsey and Victoria-British Columbia. 2005. Introduction to postgis. *Refractions Research Inc* (2005), 34–35.

[11] Joseph Redmon and Ali Farhadi. 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).

[12] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems* 28 (2015).

[13] Nicolai Wojke and Alex Bewley. 2018. Deep Cosine Metric Learning for Person Re-identification. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 748–756. https://doi.org/10.1109/WACV.2018.00087

[14] Ioannis Xarchakos and Nick Koudas. 2019. Svq: Streaming video queries. In *Proceedings of the 2019 International Conference on Management of Data*. 2013–2016.

[15] Zhuangdi Xu, Gaurav Tarlok Kakkar, Joy Arulraj, and Umakishore Ramachandran. 2022. EVA: A Symbolic Approach to Accelerating Exploratory Video Analytics with Materialized Views. In *Proceedings of the 2022 International Conference on Management of Data*. 602–616.

[16] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Fucheng Weng, Zehuan Yuan, Ping Luo, Wenyu Liu, and Xinggang Wang. 2022. Bytetrack: Multi-object tracking by associating every detection box. In *Proceedings of ECCV 2022*. Springer, 1–21.