SketchQL Demonstration: Zero-shot Video Moment Querying with Sketches

Renzhi Wu*
Georgia Institute of Technology
renzhiwu@gatech.edu

Pramod Chunduri*
Georgia Institute of Technology
pramodc@gatech.edu

Dristi J Shah
Georgia Institute of Technology
dshah371@gatech.edu

Ashmitha Julius Aravind
Georgia Institute of Technology
asjula@gatech.edu

Ali Payani
Cisco
apayani@cisco.com

Xu Chu
Georgia Institute of Technology
xu.chu@cc.gatech.edu

Joy Arulraj
Georgia Institute of Technology
arulraj@gatech.edu

Kexin Rong
Georgia Institute of Technology
krong@gatech.edu

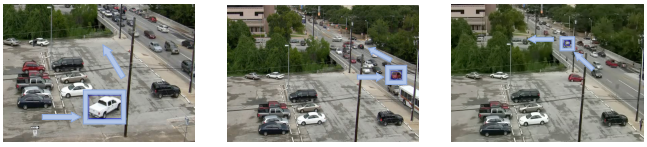
ABSTRACT

In this paper, we will present SketchQL, a video database management system (VDBMS) for retrieving video moments with a sketch-based query interface. This novel interface allows users to specify object trajectory events with simple mouse drag-and-drop operations. Users can use trajectories of single objects as building blocks to compose complex events. Using a pre-trained model that encodes trajectory similarity, SketchQL achieves zero-shot video moments retrieval by performing similarity searches over the video to identify clips that are the most similar to the visual query. In this demonstration, we introduce the graphic user interface of SketchQL and detail its functionalities and interaction mechanisms. We also demonstrate the end-to-end usage of SketchQL from query composition to video moments retrieval using real-world scenarios.

PVLDB Reference Format:
Renzhi Wu, Pramod Chunduri, Dristi J Shah, Ashmitha Julius Aravind, Ali Payani, Xu Chu, Joy Arulraj, and Kexin Rong. SketchQL Demonstration: Zero-shot Video Moment Querying with Sketches. PVLDB, 17(12): 4429 - 4432, 2024.
doi:10.14778/3685800.3685892

1 INTRODUCTION

Video moment retrieval is an important task in video analytics whose goal is to search for target moments (sequences of frames) within a video. This task has numerous applications in traffic surveillance, sports analytics, and autonomous driving. For example, transport researchers are interested in retrieving different instances of left-turning vehicles from surveillance video streams to analyze driving behaviors and improve traffic safety [1]. However, accurately detecting "left turn" motions in diverse, real-world videos can be quite challenging. Consider the illustrative video clips from a parking lot surveillance camera in Figure 1. In Figure 1a, a car begins driving towards the right side of the screen and makes a left turn towards the top, while in Figure 1c, another car starts driving towards the top of the screen and makes a left turn towards the left side. Furthermore, due to the camera's varying position and angle relative to vehicles, the turning angles might appear different on camera: the car in Figure 1b has an acute turning angle, while the car in Figure 1c has an obtuse angle. Ideally, the left turn query should capture all types of left-turn events, irrespective of the vehicle's initial direction or turning angle. Although the camera is stationary in this example, it could still subject to movements caused by environmental factors like wind or vibrations; other video streams, such as those in sports, often feature moving and panning cameras. These further complicates the identification of relevant events.



(a) Nearby car, acute angle, turning top. **(b) Distant car, acute angle, turning top.** **(c) Distant car, obtuse angle, turning left.**

Figure 1: Diverse left-turn behaviors in a traffic surveillance video.

Limitations of Current Approaches. The two main types of query interfaces for video moment retrieval, natural language-based and SQL-based, suffer from limitations in generalizability or ease-of-use and cannot adequately address the above challenges.
(1) Natural language-based interfaces retrieve target video clips based on user-specified text (*e.g.*, "Car making a left turn") and are popular within the machine learning community [6]. These methods are easy to use for non-experts, and are typically implemented by training end-to-end deep learning models that map text to raw video frames [11]. A key limitation of these interfaces is their reliance on large training data to achieve accurate retrieval [2], which limits their application outside the original training contexts.
(2) SQL-based query interfaces, predominantly developed within the data management community [3, 5, 10], support rule-based selection of clips using SQL-like syntax. They are often built upon low-level primitives extracted by pre-trained models, such as pre-trained object detectors [5, 10], object tracking models [5, 10], or

*The first two authors contributed equally to this paper.

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685892

scene graph extraction models [4]. The main advantage of SQL-based interfaces is their ability to *generalize* across different datasets and video domains with few or no labeled examples, thanks to the robust performance of pre-trained models [8]. However, SQL-base interfaces require considerable query specification time from users because translating a semantically meaningful event (*e.g.*, left turns) into SQL-like rules on top of low-level primitives (*e.g.*, location and angle of bounding boxes) can be challenging.

Video Moment Querying with a sketch-based interface. To address the above limitations, in our recent work [9], we developed SketchQL, a video database management system (VDBMS) for offline, exploratory video moment retrieval that is easy to use and generalizes well across video datasets. To improve ease-of-use, SketchQL features a *visual query interface* that enables users to sketch complex visual queries through intuitive drag-and-drop actions. By leveraging a human's inherent ability to capture complex events via sketches, SketchQL improves the usability and expressivity of query specifications for non-expert users. To improve generalizability, SketchQL operates on object-tracking primitives that are reliably extracted across datasets using pre-trained models [8]. The sketch-based query is executed by comparing the query trajectories provided by the user to object bounding box trajectories extracted using pre-trained object trackers. We developed a transformer-based neural network model that learns a similarity measure between trajectories that is robust to differences in camera perspectives and movements. SketchQL trains the model on a diverse dataset generated with a novel simulator, that enhances its accuracy across a wide array of datasets and queries.

In this paper, we introduce the user interface of SketchQL. We demonstrate the main functionalities of SketchQL (composing visual queries and retrieving similar clips) with end2end scenarios.

2 SYSTEM OVERVIEW

SketchQL consists of three key components: (1) Sketcher features a *visual query interface* that enables users to sketch complex queries through simple drag-and-drop actions, (2) Matcher compares the query trajectories provided by the user to object bounding box trajectories extracted using pre-trained object trackers and (3) Tuner is an optional component that incorporates explicit user feedback when provided to improve the retrieval quality.

SketchQL is designed to operate on top of per-frame object bounding boxes rather than raw pixels, similar to Miris [3] and STAR Retrieval [4]. Bounding box sequences for objects across frames are obtained in a preprocessing step using pre-trained object trackers [12] without dataset-specific retraining.

2.1 Sketcher: Composing visual queries

The Sketcher is the query interface of SketchQL, and it has two major components: (a) *The Canvas.* This is a whiteboard where users can place and drag objects to compose clips (Figure 2, top). (b) *The Trajectory Panel.* This is a panel where users can adjust multiple trajectories of the same object and align the trajectories of different objects (Figure 2, bottom). The Sketcher is developed on top of a popular canvas library `tldraw` (https://github.com/tldraw/tldraw) We detail the specific functionalities of Sketcher in the following.

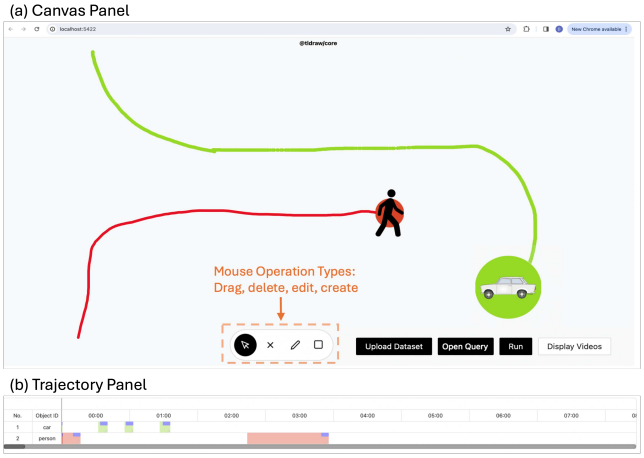
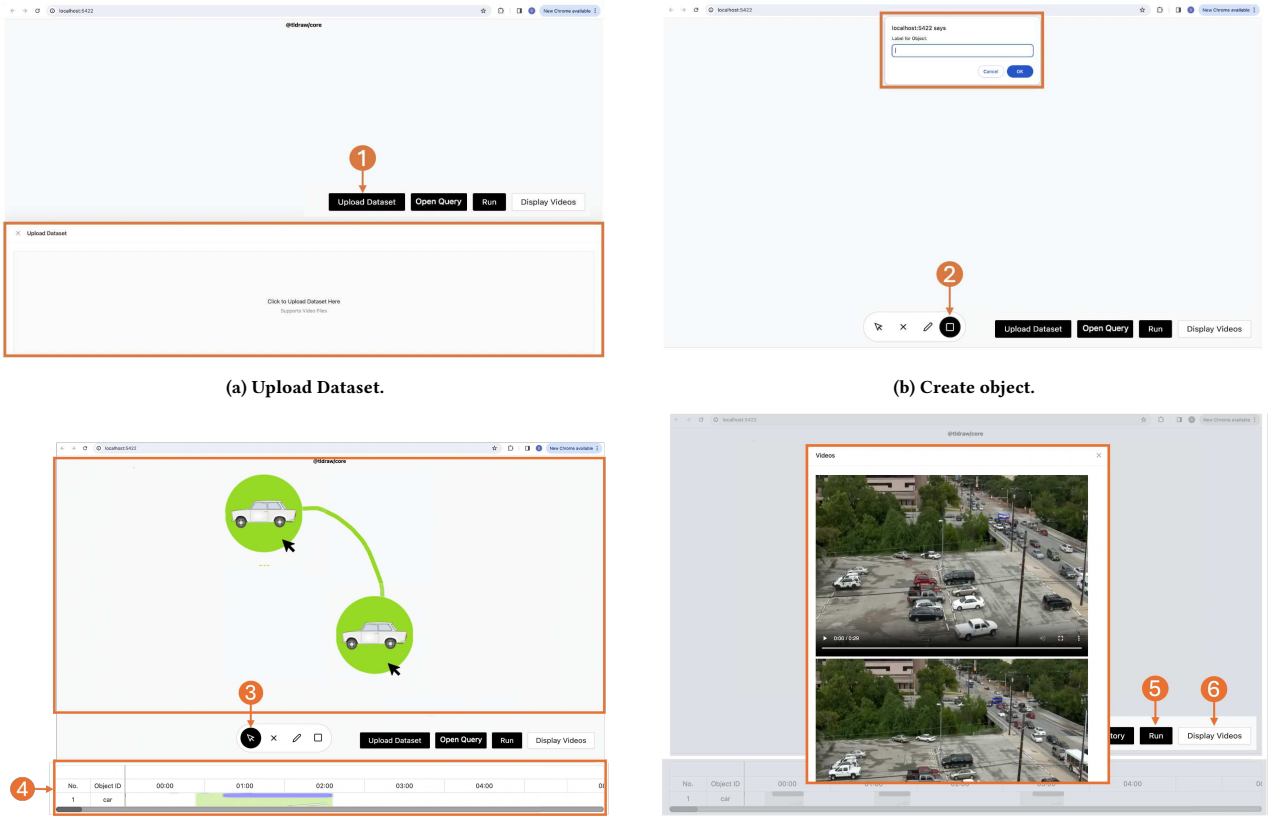


Figure 2: User interface of SketchQL.

When using the canvas, the user can select one of four types of mouse operations: drag, delete, edit, and create by clicking the corresponding buttons (shown in the pink dashed box in Figure 2). The four operations enable the following functionalities:
(1) **Object Creation**. This action allows users to select an object type and place the object on the canvas. When the user clicks on the "square" icon to select the mouse operation type as "create", an input box pops up for the user to specify the object type. Currently, about eighty common object types (*e.g.*, car, person) are supported. The user can also set a generic type *Any* that represents any types of objects. After setting the object type, the user can place one or more objects on the canvas.
(2) **Object Deletion and Editing.** The user is allowed to delete or edit an object on the canvas. When the user clicks on the "cross" icon, the mouse enters deletion mode and the user can click on an object to delete it. Similarly, when the user clicks on the "pencil" icon, the user can click on an object to change its object type.
(3) **Trajectory Creation with Drag and Drop**. The user can create trajectories when the mouse is in "Drag" mode which activates when the "cursor" icon is clicked. In this mode, the user can move an object on the Canvas via mouse drag-and-drop operations, and all the coordinates of the movements are automatically recorded. Each drag-and-drop operation is represented as a box in the trajectory panel. For example, the car movement in Figure 2 is created by three drag-and-drop operations: left turn, going straight, and right turn, so there are three boxes shown in the trajectory panel for the car object. This abstraction allows users to use each drag-and-drop operation as a building block to compose complex motions for an object, even enabling complex events involving multiple objects.

The Trajectory Panel is similar to the soundtrack panel in existing audio editing tools. It allows for the following functionalities:
(1) **Trajectory Editing**. The user can delete the boxes in the trajectory panel for each object to remove unwanted trajectories. The user can resize the box to make it shorter or longer to speed up or slow down the corresponding trajectory. The user can also rearrange the boxes to change the order of the trajectories. For example, in Figure 2, the car's initial trajectory is a left turn, followed by straight motion, and then a right turn. By reordering the three



(a) Upload Dataset.

(b) Create object.

(c) Create and Edit Trajectory.

(d) Run query and display results.

Figure 3: SketchQL end-to-end usage demonstration

boxes, users can edit the event to depict a different sequence, such as a left turn, a right turn, and then straight motion.

(2) **Trajectory Coordination Among Multiple Objects.** When creating events with multiple objects, we need to coordinate the timing of the trajectories. For example, in the event depicted in Figure 2, the motion that the car goes right happens first, and after a long time the person goes right. To create a synchronized event in which the person and the car go right at the same time, we can move the second box of the person object in the trajectory panel to align with the second box of the car object.

Apart from the canvas panel and the trajectory panel, the interface also provides a few buttons in the bottom right corner as shown in Figure 2. The buttons provide the following functionalities: (1) *Dataset Uploading*. When the user clicks on the button "Upload Dataset", a window pops out allowing the user to select a video file to upload. After a dataset is uploaded, future queries will be executed using this dataset. (2) *Query replay*. When the user clicks on the "Open Query" button, a window pops out displaying an video that animates the defined event, so the user can check the query holistically, and optionally go back to editing the query. (3) *Query execution*. When the user clicks on the button "Run", the visual query is sent to backend to be executed, where Matcher is invoked to find similar video clips. (4) *Display query results*. When the user clicks on the button "Display Videos", a window pops out listing the found similar video clips.

2.2 Matcher: Identifying Similar Clips

The Matcher is the backend of SketchQL. We discuss it briefly here, and more details can be found in our research paper [9].

The Matcher identifies video clips C_V that are most similar to a given visual query C_Q through sliding-window similarity search. The core challenge is defining the similarity function $sim(C_Q, C_V)$ that is robust to camera angles and noises. We developed a pre-trained model that encodes the bounding box trajectories in C_Q or C_V as embeddings, and then similarity is measured as the cosine similarity between the embeddings. The model architecture is a transformer that encodes the the sequences of bounding box trajectories of multiple objects into one embedding vector.

To train the embedding model, we propose synthesizing labeled training data using a custom trajectory simulator, inspired by the wide adoption of simulators in generating training data for autonomous driving models. The simulator operates on top of the bounding box abstraction, enabling it to synthesize trajectories across video domains. The high-level idea of our simulator is to generate motions in a 3D space and create 2D video clips by recording the event from virtual cameras placed at random locations in the 3D space. Intuitively, 2D video clips from the different cameras of the same 3D clip are positive (similar) examples, and 2D video clips from different 3D clips are negative (dissimilar) examples.

SketchQL also has an optional Tuner component that can adapt the learned similarity measure according to user feedback. Since SketchQL provides decent results without needing user feedback, in this paper we focus on demonstrating the zero-shot retrieval capability of SketchQL and omit the discussion on Tuner.

3 DEMONSTRATION

We demonstrate the usage of SketchQL with an end-to-end scenario. We use a real-world traffic surveillance dataset [7] that is used extensively in the computer vision community. We consider two common queries: (Q1) a car making a left turn and (Q2) car and person moving perpendicularly to each other. Q1 only has one object, while Q2 involves two objects. The overall pipeline of Q2 is similar to that of Q1; it only differs in how to coordinate the two objects. Therefore, we first illustrate the overall pipeline with Q1, and then we discuss how to coordinate multiple objects with Q2.

3.1 End-to-end Demo with Q1

The end-to-end usage of SketchQL is illustrated in Figure 3.

Step 1: Uploading dataset and initialization. The user uploads a video by clicking the "Upload Dataset" button and selecting a video file in the pop-up window. After this, SketchQL initializes by extracting the object tracking primitives for the dataset.

Step 2: Object Creation. The user clicks on the creation icon, *i.e.*, the "square" icon, sets the object type to "Car" in the pop-up input window, and then clicks on the canvas to place the "Car" object.

Step 3: Trajectory Creation. The user clicks the "cursor" icon to enable the mouse's "Drag" mode. Then, the user drags the "Car" object on the canvas to make a left turn. The user drops the "Car" object when the left turn finishes.

Step 4: Trajectory Editing. The user clicks on the "Open Query" button to replay the event just created. The user can edit the trajectory if needed. For example, the user may delete the trajectory from the Trajectory Panel and then drag the object to create a new trajectory, If the user is satisfied with the shape of the trajectory but hopes to let the car make a left turn faster, the user can stretch the corresponding box on the Trajectory Panel to shorten the box.

Step 5: Query Execution. The user clicks on the "Run" button. The visual query is then sent to the backend to be executed.

Step 6: Checking the Found Video Clips. After query execution, the user can click "Display Videos" to see the list of similar video clips found by the Matcher sorted by their query similarity scores.

3.2 Multi-object Event Query Demo with Q2

We demonstrate how to create queries with multi-object events with a simple query, car, and person moving perpendicularly to each other. Step 1, Step 5, and Step 6 are the same as in Section 3.1, and we explain Step 2, Step 3, and Step 4.

Step 2 (Multi-object): Object Creation. The user follows Step 2 in Section 3.1 to create one "Car" object and one "Person" object.

Step 3 (Multi-object): Trajectory Creation. The user clicks on the "cursor" icon to enable the mouse's "Drag" mode. The user drags the "Person" object to move horizontally. After that, the user drags the "Car" object to move vertically. With this creation, the "Person" object moves first, and the "Car" object moves after. To make them move simultaneously, next we coordinate their movements.

Step 4 (Multi-object): Trajectory Editing. In the Trajectory Panel, the user drags the box representing the trajectory of the "Car" object to the left to ensure it synchronizes with the box representing the trajectory of the "Person" object. Figure 4 shows the Trajectory Panel after synchronization (originally the box for the "Car" object is on the right side of the box for the "Person" object).

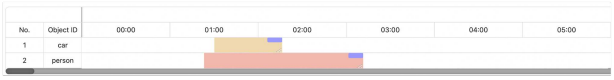


Figure 4: Trajectory panel for multi-object query Q2 after Step 4.

4 CONCLUSION

In this paper, we demonstrated SketchQL, a video database management system (VDBMS) for retrieving video moments with a visual query interface. SketchQL's interface allows query specification with simple mouse drag-and-drop operations. The interface also supports creating query with multiple objects.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under grants IIS-2335881 and IIS-2238431.

REFERENCES

[1] Osama Abdeljaber, Adel Younis, and Wael Alhajyaseen. 2020. Analysis of the trajectories of left-turning vehicles at signalized intersections. *Transportation research procedia* 48 (2020), 1288–1295.

[2] Lisa Anne Hendricks, Oliver Wang, Eli Shechtman, Josef Sivic, Trevor Darrell, and Bryan Russell. 2017. Localizing moments in video with natural language. In *Proceedings of the IEEE international conference on computer vision*. 5803–5812.

[3] 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.

[4] Yueting Chen, Nick Koudas, Xiaohui Yu, and Ziqiang Yu. 2022. Spatial and temporal constrained ranked retrieval over videos. *Proceedings of the VLDB Endowment* 15, 11 (2022), 3226–3239.

[5] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. BlazeIt: optimizing declarative aggregation and limit queries for neural network-based video analytics. *VLDB* 13, 4 (2019), 533–546.

[6] Meng Liu, Liqiang Nie, Yunxiao Wang, Meng Wang, and Yong Rui. 2023. A survey on video moment localization. *Comput. Surveys* 55, 9 (2023), 1–37.

[7] Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, JK Aggarwal, Hyungtae Lee, Larry Davis, et al. 2011. A large-scale benchmark dataset for event recognition in surveillance video. In *CVPR 2011*. IEEE, 3153–3160.

[8] Shuai Shao, Zeming Li, Tianyuan Zhang, Chao Peng, Gang Yu, Xiangyu Zhang, Jing Li, and Jian Sun. 2019. Objects365: A large-scale, high-quality dataset for object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*. 8430–8439.

[9] Renzhi Wu, Pramod Chunduri, Ali Payani, Xu Chu, Joy Arulraj, and Kexin Rong. 2025. SketchQL: Video Moment Querying with a Visual Query Interface. In *Proceedings of the 2025 ACM SIGMOD International Conference on Management of Data*.

[10] 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.

[11] Songyang Zhang, Houwen Peng, Jianlong Fu, and Jiebo Luo. 2020. Learning 2d temporal adjacent networks for moment localization with natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 12870–12877.

[12] 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 *ECCV*. Springer, 1–21.