# SketchQL: Video Moment Querying with a Visual Query Interface

## ABSTRACT

Localizing video moments based on the movement patterns of objects is an important task in video analytics. Existing video analytics systems offer two types of querying interfaces based on natural language and SQL, respectively. However, both types of interfaces have major limitations. SQL-based systems require high query specification time, whereas natural language-based systems require large training datasets to achieve satisfactory retrieval accuracy.

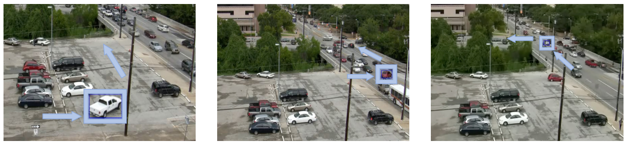
To address these limitations, we present SketchQL, a video database management system (VDBMS) for offline, exploratory video moment retrieval that is both easy to use and generalizes well across multiple video moment datasets. To improve ease-of-use, SketchQL features a *visual query interface* that enables users to sketch complex visual queries through intuitive drag-and-drop actions. To improve generalizability, SketchQL operates on object-tracking primitives that are reliably extracted across various datasets using pre-trained models. We present a learned similarity search algorithm for retrieving video moments closely matching the user's visual query based on object trajectories. 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. We evaluate SketchQL on four real-world datasets with nine queries, demonstrating its superior usability and retrieval accuracy over state-of-the-art VDBMSs.

**ACM Reference Format:**
. 2024. SketchQL: Video Moment Querying with a Visual Query Interface. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

*Video moment retrieval*, also known as *video moment localization*, is an important task in video analytics whose goal is to search for target moments (where each moment is a sequence of frames) within a video. This task has numerous applications in traffic surveillance [51], sports analytics [25], and autonomous driving [55, 59]. 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 [5].

There are two main types of query interfaces for video moment retrieval, SQL-based and natural language-based, primarily developed by the data management and machine learning communities respectively. However, it is challenging to use these interfaces to effectively retrieve even simple events such as "car making a left-turn" in diverse, real-world videos, as shown in Figure 1.



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

**Figure 1: Challenges in Video Moment Retrieval – Diverse left-turn motions in a traffic surveillance video stream [51].**

**1. SQL-based Interface.** Many recent video database management systems (VDBMSs) use a SQL-based interface [9, 12–14, 21, 34, 40, 52, 54] to retrieve relevant video moments. These systems either specify object and action categories of interest as query predicates [9, 14, 34, 52, 54], or apply spatio-temporal rules over video frames [12, 13]. They utilize low-level primitives extracted using pre-trained models such as pre-trained object detectors [34, 54], object tracking models [34, 54], or scene graph extraction models [12]. 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 for extracting low-level primitives [48].

However, SQL-based systems require considerable time for query specification, as it is often non-intuitive to translate visual patterns into SQL. For example, the left-turn event in Figure 1c can be retrieved using the following SQL query, which uses bounding boxes extracted from an object tracking model as the low-level primitives:

```
Q1: SELECT car FROM (PROCESS InputVideo
        PRODUCE car USING ObjectTracker)
    WHERE TurningAngle(car) > 15 deg
    AND RelativeXVelocity(car) < 0
    AND MovingDistance(car) > 20
```

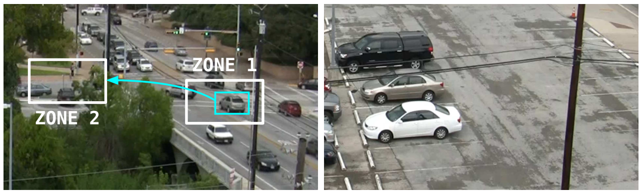
The query enforces the following conditions: (1) The turning angle must exceed 15 degrees, confirming that the vehicle is making a turn; (2) The x-component of the velocity vector should be negative, which helps distinguish left turns from right turns; (3) The vehicle must move at least 20 pixels, eliminating false positives caused by minor camera shakes. Each condition requires a non-trivial implementation of user-defined functions (UDFs), such as `TurningAngle` shown in Listing 1. Moreover, users must manually adjust the query parameters to match the diverse left-turn instances showcased in Figure 1. Due to these complexities, our experiments find that even simple queries like identifying left-turns can consume a significant amount of programming time (exceeding 10 minutes).

There are certain workarounds to simplify query specification for SQL-based methods, which come with other compromises. For example, Miris [9], a state-of-the-art SQL-based VDBMS, identifies left turns using zonal markings in fixed camera configurations,

*Conference'17, July 2017, Washington, DC, USA*
© 2024 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

```
def TurningAngle(boxes, angle):
    centroids = _bounding_box_centroid(boxes)
    # get bounding box centroid movements between frames
    centroids_rel = [centroids[i+1]-centroids[i] for i in np.
      arange(0, len(centroids)-1, 10)]
    # find the angle of the relative velocity vector
    for i in range(len(centroids_rel) - 1):
        u1 = centroids_rel[i]/np.linalg.norm(centroids_rel[i])
        u2 = centroids_rel[i+1]/np.linalg.norm(centroids_rel[i+1])
        angle = np.arccos(np.clip(np.dot(u1, u2), -1.0, 1.0))
        angles.append(np.degrees(angle))
    # return frames that satisfy the predicate
    return [angles[i] > angle for i in range(len(angles))]
```

**Listing 1: TurningAngle function for identifying left turns.**



**(a) Miris [9] requires hardcoded zonal markings to detect a left turn.**

**(b) Clip [43] returns frames with parked cars facing "left".**

**Figure 2: Limitations of SoTA VDBMSs – Illustrative video moments retrieved by VDBMSs for "car making a left turn" query.**

as shown in Figure 2a. However, this simplified approach only works with fixed cameras and simple movement patterns, unable to accurately answer several queries in our evaluation.

**2. Natural language-based Interface.** 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 ML community [11, 22, 38, 53, 61, 63]. The main advantage of these methods is that they are *easy to use* for non-experts. These methods are typically implemented by training end-to-end deep learning models that directly map text to raw video frames [62, 63].

A key limitation of these methods is their requirement of large training dataset to support accurate retrieval [8, 26, 44], which limits their application outside the original training contexts. For example, ActivityNet, a popular benchmark for human activities in videos, requires ~8 Amazon Mechanical Turk workers to annotate each video [10, 27]. Despite this immense labeling effort, we discover that a retrieval model trained on ActivityNet struggles to identify even a single player-kicking-ball event in soccer game videos. Researchers have also proposed large pre-trained vision-language models for zero-shot video analysis [43]. These models also suffer from the generalization problem and often require additional fine-tuning to perform well on other datasets [43, 65, 66]. For example, when prompted with the natural language query *"A car makes a left turn"*, the pre-trained vision-language model Clip [43] mostly retrieves frames containing stationary *cars* facing *left*, as shown in Figure 2b.

As a result, these natural language-based methods are primarily used in closed-world scenarios such as indoor activities [22], where clips are retrieved only from a pre-defined domain with a large collection of labeled training data.

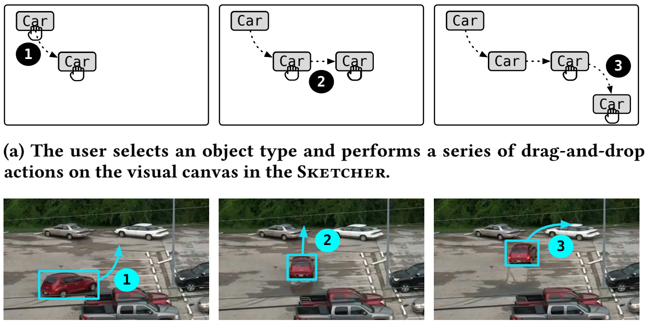
**Challenges.** Table 1 summarizes the advantages and limitations of SQL-based and natural language-based interfaces. Systems with SQL-based interfaces have better generalizability across datasets but are not easy to use; systems with natural language interfaces are easy to use but do not generalize to datasets different from the training set. A VDBMS must ideally satisfy both criteria:

| Query Interface | Representative Systems | Ease-of-use | Generalizability |
|---|---|---|---|
| SQL | Miris [9] | Low | **High** |
| Natural language | Clip [43], 2DTAN [63] | **High** | Low |
| Visual | SketchQL (this paper) | **High** | **High** |

**Table 1: Comparison of different video moment query interfaces. Along each dimension, bold text denotes the best setting.**

*1. Ease-of-use.* The system must allow intuitive expression of intricate object motions and multi-object interactions. It should automatically retrieve relevant video moments without significant programming time and manual parameter tuning.

*2. Generalizability.* The system should generalize across datasets and trajectory variations without extensive real-world training data. It must support arbitrary trajectory patterns and achieve high accuracy even for complex multi-object trajectories.



**(a) The user selects an object type and performs a series of drag-and-drop actions on the visual canvas in the Sketcher.**

**(b) The Matcher automatically maps the user sequence to the relevant video moment using learned similarity search.**

**Figure 3: – Illustration of SketchQL's moment retrieval system**

**3. Our Approach – Sketch-Based Interface.** In this paper, we present SketchQL, a VDBMS for offline, exploratory video moment retrieval that is both easy to use and generalizes across datasets. SketchQL focuses on object track queries [9], which queries video moments based on object trajectories or trajectory interactions. SketchQL addresses the aforementioned challenges with its three key components.

**(1) Sketcher.** The first component, Sketcher, features a *visual query interface* that enables users to sketch complex queries through simple drag-and-drop actions. For example, Figure 3a illustrates how users can specify a query to find a car making a left turn followed by a right turn with drag-and-drop. By leveraging a human's inherent ability to capture complex events via sketches, Sketcher improves the usability and expressivity of query specifications for non-expert users. To support more complex queries involving *multiple* objects, the Sketcher also provides an intuitive composition interface (§3) to create, edit, and align sketches of multiple objects.

**(2) Matcher.** To match user-provided visual queries to video moments, SketchQL utilizes a learned similarity search module called the Matcher. This module compares the query trajectories provided by the user to object bounding box trajectories extracted using pre-trained object trackers, as illustrated in Figure 3b. It encodes similarities between trajectories using a transformer-based neural network model and outputs the top-K similar moments.

Matcher provides high generalizability via two key features:

- Similar to existing SQL-based methods (e.g., Miris [9]), the module utilizes object tracking primitives, which can be reliably extracted across datasets with pre-trained models [48].
- The neural network model for encoding similarity is pre-trained on a large, diverse set of bounding box trajectory pairs, generated automatically through a novel synthetic data generation process. Since the model is exposed to a wide diversity of trajectories beyond any single dataset, it can retrieve trajectories across datasets without manual parameter tuning.

**(3) Tuner.** The Matcher works well out-of-the-box across datasets, as we will show in experiments (§6.2). However, visual queries can be ambiguous, leading to multiple interpretations. Consider the query in Figure 3a: it is unclear whether the user is only interested in the left-turn motion or if the car's initial direction also matters. In such scenarios, refining our pre-trained transformer model (that measures the similarity between query and video) to align more closely with the user's intent can enhance result relevancy.

To address this, we develop a human-in-the-loop module called the Tuner that incorporates explicit user feedback. Specifically, the Tuner allows users to label the Matcher's outputs as positive or negative examples. It leverages these labeled examples to rapidly fine-tune the Matcher's transformer model using state-of-the-art techniques. The fine-tuned Matcher then generates updated results with better accuracy.

**Contributions.** The key contributions of this paper are:

- We develop a VDBMS to address the ease-of-use and generalizability limitations of SoTA VDBMSs, using a novel architecture consisting of Sketcher, Matcher, and Tuner (§2).
- We introduce a visual query interface, Sketcher, to query video moments of object trajectories and trajectory interactions. We illustrate that Sketcher enables users to express object track queries effectively (§3).
- We propose a learned similarity search module, Matcher, to match user sketches to real-world video moments based on object trajectories. Matcher leverages a novel synthetic training data generation method to pre-train a transformer model on diverse trajectories, thus generalizing across a wide range of datasets and queries (§4).
- We develop a human-in-the-loop module, Tuner, to accurately adapt the system to each specific query as necessary through fine-tuning the learned model based on user feedback (§5).
- We evaluate SketchQL on diverse datasets and show that it significantly improves the moment retrieval accuracy over SQL-based and natural language baselines (§6).

# 2 SYSTEM OVERVIEW

In this section, we present an overview of the key components and workflow of SketchQL.

## 2.1 The Bounding Box Abstraction

Querying raw videos requires extensive training per dataset [23, 39]. To improve generalizability, SketchQL is designed to operate on top of per-frame object bounding boxes rather than raw pixels, similar to Miris [9] and STAR Retrieval [12]. Bounding box sequences for objects across frames are obtained using pre-trained object trackers [64] without dataset-specific retraining.

Specifically, each bounding box $B = (x_l, y_l, x_h, y_h)$ is determined by two points - the top right coordinate $(x_h, y_h)$ and the bottom left coordinate $(x_l, y_l)$. A bounding box sequence $BS(t_s, t_e) = \{B_1, B_2, ...\}$ for an object consists of bounding boxes tracked over time, where $t_s$ and $t_e$ are the start and end frames. Each object is defined by its type $T$ (*e.g.*, car) and bounding box sequence from frames $t_s$ to $t_e$: $O(t_s, t_e) = (T, BS(t_s, t_e))$. We pad frames that do not contain the object with empty bounding boxes $B_{\text{emp}} = (0, 0, 0, 0)$ to create fixed length sequences between $t_s$ and $t_e$. A video clip between frames $t_s$ and $t_e$ is defined as a set of objects $C(t_s, t_e) = \{O_1(t_s, t_e), O_2(t_s, t_e), \ldots, O_m(t_s, t_e)\}$. The full video $V$ contains all the objects, with bounding boxes over the entire duration: $V = \{O_1(0, t_{\max}), \ldots\}$. Each clip in $V$ then consists of the subset of objects present in the sub-range of frames $[t_s, t_e] \subseteq [0, t_{\max}]$.

Admittedly, this bounding box abstraction loses detailed information like color and texture from the raw video. However, bounding boxes have proven useful for a wide range of applications [9] since they enable methods to work across different videos with minimal adaptation efforts. Furthermore, the videos retrieved by SketchQL still contain the original raw pixel information. This enables applying more fine-grained predicates (*e.g.*, color detection, re-identification) beyond the bounding box abstraction if needed.

## 2.2 SketchQL Workflow

The overall workflow of SketchQL is shown in Figure 4. SketchQL workflow contains four phases:

(0) **Object tracking**. This is a *one-time pre-processing* phase that converts the raw video into a set of objects $V = \{O_1(0, t_{\max}), \ldots\}$ using the bounding box abstraction (§2.1). For each object, apart from the bounding boxes, object tracking models also provide the object type information [64]. Once extracted, the resulting $V$ can be reused for future visual queries. We use the state-of-the-art object tracker ByteTrack [64] for this phase. Note that our target usage scenario is *offline* exploratory video analytics, where users interactively query over bounding box trajectories extracted from the videos to understand object movement patterns and interactions.

(1) **Creating Visual Query**. Users compose visual queries using the drag-and-drop Sketcher interface (Figure 5a). For example, to query video clips of a car turning left, moving straight, and then right, the user selects a car object, drags it in a leftward motion, then straight, and finally in a rightward motion on the canvas. This dragging motion is automatically recorded as a sequence of bounding boxes representing the car's trajectory over time (Figure 5b).

Through these simple drag-and-drop gestures, the user creates a visual query clip $C_Q$ containing the desired object motions. The key challenge is enabling the intuitive composition of complex multi-object scenes and motions. The Sketcher provides an efficient visual interface for this complex query specification (§3).

(2) **Similarity search**. For a given visual query $C_Q$, our goal is to find video clips, each represented by $C_V(t_s, t_e)$, with the highest similarity to this query.

The key challenge is measuring the similarity $sim(C_Q, C_V)$ between the bounding box sequences in the query and video clips. This measurement is challenging due to variations in camera angles

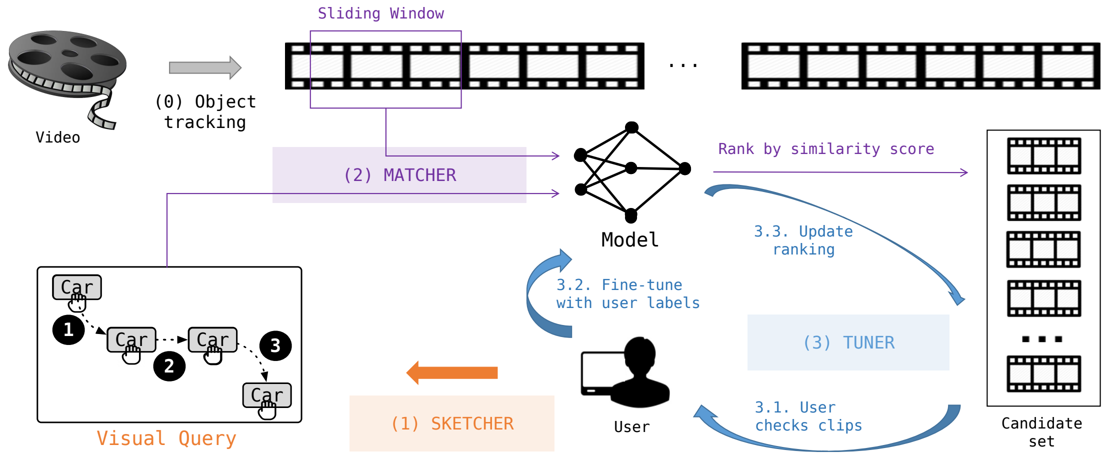


Figure 4: SKETCHQL processes moment queries in four phases - (0) Object Tracking, (1) SKETCHER, (2) MATCHER, and (3) TUNER.

and movements in real-world videos, such as shown in Figure 1. To address this, our MATCHER module proposes to pre-train a model that learns a similarity function $sim(\{BS_1, BS_2, ...\}, \{BS'_1, BS'_2, ...\})$ between any two set of bounding box sequences where $BS_i$ is the bounding box sequence of the $i^{th}$ object in query and $BS'_i$ is the bounding box sequence of the $i^{th}$ object in a video clip. The model is pre-trained on diverse synthetic trajectories designed to capture diverse real-world patterns (§4.2). Since videos can be represented as bounding box sequences across domains, our pre-trained model generalizes to real-world datasets with zero or minimal adaption, as evidenced by our experiments.

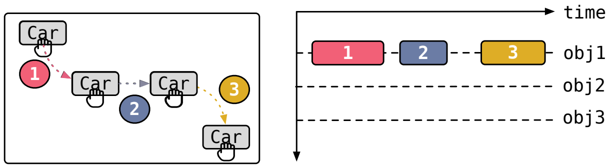
The MATCHER uses the pre-trained model to retrieve the clips via a sliding window similarity search, as shown in Figure 4. First, it identifies the candidate clips by iterating over all combinations of objects within each sliding window that matches the query object types. It then uses the pre-trained model to measure the similarity between the query bounding boxes and each candidate clip and retrieves the top $k$ clips with the highest similarity scores (§4.3).

(3) **Incorporating user feedback**. The pre-trained model in MATCHER works well across datasets. To further boost performance, we allow adapting the model specifically to each query. To support this, we introduce a TUNER module that can adapt the pre-trained model by incorporating user feedback at query time. Specifically, the TUNER enables users to provide additional feedback by labeling candidate clips as positive or negative examples. The TUNER uses these labels to fine-tune the pre-trained model to the specific query. The fine-tuned model re-predicts similarity scores for the candidate clips, re-ranking the results. Users can iteratively provide more feedback to further refine the model and rankings.

In the following sections, we present the internals of the three core components of SKETCHQL. We discuss the visual query interface, SKETCHER, in §3. We then describe the MATCHER's synthetic data generation and learned similarity search methods in §4. Finally, we discuss the TUNER's fine-tuning strategy to improve the accuracy of the MATCHER in §5.

## 3 SKETCHER: COMPOSING VISUAL QUERIES

This section describes our proposed visual query interface, the SKETCHER. In contrast to SQL's textual syntax, the SKETCHER uses a *visual query language* comprised of visual components to construct trajectories of objects and represent relationships between them. The SKETCHER has two components: (a) **The Canvas**. This is a whiteboard where users can place and drag objects to compose clips, as shown in Figure 5a. (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 (shown in Figure 5b). These visual components make it easy for users to create query clips visually by allowing the following actions:



(a) Canvas.

(b) Trajectory Panel.

Figure 5: SKETCHER Interface

(1) **Object Creation**. This action allows users to select an object type (*e.g.*, through a drop-down menu) and place the object on the canvas. Users can create an object of a specific type (*e.g.*, car) or an *Any* type that encompasses all types of objects that the object detection model can identify.

(2) **Trajectory Creation with Drag and Drop**. When a user moves an object on the Canvas via drag-and-drop, all the coordinates of the movements are automatically recorded as bounding box sequences $BS$. This way, both the query clip and the video clips are represented in the same bounding box format. We record trajectory as it is with possible noise and design the MATCHER to be noise-tolerant.

(3) **Trajectory Editing**. To enable users to compose complex motions intuitively, the Trajectory Panel allows users to edit and compose multiple trajectories (of multiple objects). In other words, users do not have to create a complete query in one pass and can create a query one segment by segment. The Trajectory Panel is similar to the soundtrack panel in existing audio editing tools.

Consider a scenario where the user wants to identify clips in a traffic surveillance video stream where a car first takes a left turn, then goes straight, stops for a while, and then takes a right turn (Figure 5a). To achieve this, the user first creates a car object on Canvas. The user then drags the car object to make a left turn (indicated by the red dashed arrow in Figure 5a) to specify the query trajectory. Once the object is released from the dragging motion, the trajectory is automatically recorded and appears on $obj1$'s timeline in the Trajectory Panel (The red "1" box in Figure 5b). Similarly, the user creates a second trajectory by dragging the car object in a straight line (blue dashed arrow in Figure 5a), resulting in a second trajectory box on the timeline. Users can shift these trajectory boxes on the timeline to change the ordering of the two events or to adjust the duration of the car's stationary period. Additionally, users can also adjust the length of the trajectory boxes along the timeline to change the speed of the movements. For complex queries involving multiple objects, users can repeat these steps. For example, they can add a pedestrian object, set its trajectory, and adjust the trajectory boxes of the pedestrian and the car to define temporal correlations between them in the Trajectory Panel.

**Query Compilation.** Query compilation converts the object trajectories created on the Canvas into a query clip $C_Q(0, t_e) = \{O_1, O_2, ..., \}$ (defined in §2.1) by recording the bounding box sequences for each object's trajectory.

**Limitations of Visual Querying.** Due to the inherent nature of a visual interface, visual querying is best suited for queries that can be expressed by sketches, such as arbitrary movement patterns of objects. Currently, the unsupported types of queries include:

1. summarization queries – *e.g.*, count number of cars in a frame
2. queries with hard constraints – *e.g.*, Speed(car) > 30 km/h or Speed(car) is larger at one part of trajectory than another part.
3. semantic queries – *e.g.*, person laughing

It is possible to extend visual querying to support type 2 and type 3 queries by adding additional annotations (*e.g.*, velocity, activity class) to the objects, which we leave as future work.

## 4 MATCHER: IDENTIFYING SIMILAR CLIPS

The Matcher aims to identify video clips $C_V$ that are most similar to a given visual query $C_Q$. The pseudo-code for the similarity search algorithm is shown in Algorithm 1.

We perform a sliding window similarity search where the default window size is the duration of the query. As indicated in Line 4, we vary the window size by different scales to consider candidate clips with a different duration from the query. At Line 6, we iterate over all possible clips (combinations of objects) of sub-video beginning at $t_s$ and ending at $t_e$[1]. For each clip $C_V$, we check whether it is a valid candidate that contains the same number of objects and the same object types as the query $C_Q$ at Line 7. For a valid candidate clip, we obtain its similarity score with the query using a similarity function $sim(C_Q, C_V)$. Finally, we handle temporally overlapping clips with identical objects from Line 9 to Line 14. For instance, if one clip spans from time [0, 10] and another similar clip from [1, 11], only the clip with the highest score is retained. Optionally, SketchQL also allows filtering candidates with user-defined functions (UDFs) in a pre-processing step before the similarity search or a post-processing step after the search to support semantic queries and queries with hardcoded parameters.

**Algorithm 1:** Similarity search

```
Input: Query $C_Q$, Video $V$, similarity function $sim$, number of
       clips $k$ to be retrieved
Output: top $k$ clips with highest similarity scores
1  Let $t_w$ be the duration of $C_Q$ and $t_{\max}$ be the length of $V$.
2  Let $S$ be the set of candidate clips.
3  for $t_s \leftarrow 0$ to $t_{max}$ do
4      for scale in [0.5, 0.75, 1, 1.25, 1.5, 2] do
5          $t_e = t_s + t_w$*scale
6          for $C_V$ be a clip with subset of objects in $V(t_s, t_e)$ do
7              if $C_V$ has same num and type of objects as $C_Q$ then
8                  score = $sim(C_Q, C_V)$
9                  if objects in $C_V$ makes up a clip $C'_V$ in $S$ then
10                     if $C_V$ and $C'_V$ have no temporal overlap then
11                         $S$.append($C_V$)
12                     else if score of $C_V$ is higher then
13                         replace $C'_V$ with $C_V$ in $S$.
14                     end
15                 else
16                     $S$.append($C_V$)
17                 end
18                 if len(S)>k then
19                     Pop the clip with the smallest score in $S$
20                 end
21             end
22         end
23     end
24 end
25 Optional: filter $S$ with UDFs.
26 return $S$
```

### 4.1 Challenge: Measuring Similarity

The core objective in similarity search is defining the similarity function $sim(C_Q, C_V)$. For our target object track queries, variable camera perspectives and movements pose a major challenge: for example, the same moving objects can have different trajectories and sizes when recorded by cameras from different angles as illustrated in Figure 6. Ideally, our similarity function must remain unaffected by camera angles since users may describe object movements from perspectives different from the video's actual viewpoint. Another challenge is the camera movements caused by environmental factors such as wind and vibration. Users might assume a stationary camera when specifying queries, but SketchQL needs to identify relevant events despite minor camera movements.

**Baseline: Classic Distances on Manual Features.** A naïve solution is to use classic distance metrics like Euclidean distance and DTW distance [57] to measure the similarity on manually extracted features (*e.g.*, position and angle of objects.), which reduces the problem into a relational time series query problem. However, this approach does not work well, as we verify in our experiments in §6.4. We observe that the classic distance metrics are sensitive to camera angles and noises like camera movements.

Instead, we propose to train an end-to-end model that takes in $C_Q$ and $C_V$ as input and outputs a similarity score. The model encodes the trajectory similarities into distances in the learned embedding space, thus obviating the need for manual feature engineering. The learned model is also trained to ensure the similarity is robust to

[1] Note that this step is just for conceptual illustration; there are more efficient ways of implementing this without requiring iterating over all the clips.

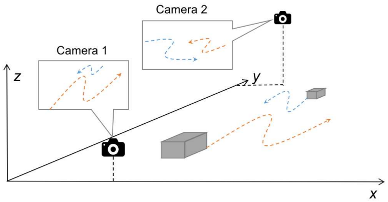


**Figure 6: Camera perspective variations - the same trajectory can appear different when recorded from different camera angles.**

camera angles and noises. We discuss how to obtain training data for the model in §4.2 and how to design an effective architecture and training procedure in §4.3.

## 4.2 Scalable Training Data Generation

Obtaining labeled training data is challenging for two reasons: (1) A large amount of labeled data is needed to train an accurate model, and (2) For the trained model to work across domains out of the box, the training data must be as diverse and generic as possible. Inspired by the wide adoption of simulators in generating training data for autonomous driving models [19, 47], we propose synthesizing labeled training data using a custom trajectory simulator. 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. We also insert random variations (e.g., flipping, shifting, and white noise) into the bounding box sequences of positive examples to simulate real-world variations. We do not consider object types during training data generation, as clips whose object types do not match those of the query are automatically skipped in Algorithm 1 (Line 7).

**3D-Clip Generation.** To generate a 3D clip with a single object represented as a 3D box, we use the following method. For clips with multiple objects, this process is repeated. The object's dimensions–width, length, and height–are uniformly sampled from (0,0.1). Note that these dimensions may appear differently in a 2D video representation based on the object's distance from the camera. The object is placed at $(x_0, y_0, 0)$ in a 3D space, where $x_0$ and $y_0$ are randomly sampled from $[0, L_{\max}]$, with $L_{\max} = 2$ being a reference scale for the object trajectory.

We make the object perform a random walk in the x-y plane over a frame count $n_{\text{frame}}$, which is randomly sampled from $[100, 1000]$. Since real-world objects do not frequently change their velocities, we randomly select $n_{\text{change}}$ (sampled from $[0, 10]$) frames where the object changes its velocity while the velocity remains constant at all other frames. The object's x and y velocity at each frame are sampled from $[-L_{\max}/100, L_{\max}/100]$, so that the trajectory is at the scale of $[L_{\max}, 10L_{\max}]$.

To obtain the corresponding 2D clip from the 3D clip, a pin-hole projection is performed based on the camera's location and orientation [15]. This projection retains expected camera characteristics, such as size scaling with proximity. The 3D box is then projected into a 2D polygon, which is then enclosed by a minimum bounding box to create sequences of 2D bounding boxes for each object.

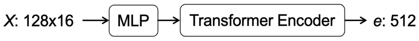


**Figure 7: Encoder architecture for trajectory representations.**

**Positive & Negative Example Generation.** Users may create visual queries based on one perspective while the actual video is recorded from another angle. Despite these differing viewpoints, both clips originate from the same 3D event (Figure 6) and should therefore have a similarity score of 1. Leveraging this insight, we propose to generate positive and negative training examples as follows: pairs of 2D clips recorded by different cameras serve as positive examples, and pairs of clips capturing different events are negative examples. We ensure the two clips in one negative pair have the same number of objects to avoid trivial negative examples.

To further improve the robustness of the trained model, we inject the following random variations into each clip:

- random flipping horizontally or vertically;
- random rotation by a random angle;
- random white noise on the coordinates at each frame;
- randomly dropping $[0, 20\%]$ frames in the beginning and end;
- randomly warping the frames, e.g., $[\text{f}_1, \text{f}_2, \text{f}_3, \text{f}_4, \text{f}_5, \text{f}_6]$ becomes $[\text{f}_1, \text{f}_3, \text{f}_5, \text{f}_5, \text{f}_6, \text{f}_6]$ where we speed up the first half and slow down the second half.

We apply each variation to the clip with a 50% probability. For variations with parameters (*e.g.*, rotation angle), parameters are sampled randomly.

## 4.3 Learned Similarity Function

Our goal is to train an encoder model that effectively learns object trajectory representations. The model first encodes the input $X_V$ (and $X_Q$) into an embedding vector $e_V$ (and $e_Q$). The similarity between the query and a video clip is obtained as the cosine similarity between their embedding vectors: $sim(C_Q, C_V) = cos(e_Q, e_V)$.

Figure 7 shows our proposed encoder architecture. An MLP first encodes object trajectories in each video frame into a lower-dimensional embedding vector of dimension 128. These embeddings, along with the positional encoding of the frame position, are fed into a transformer encoder to encapsulate spatial-temporal information across frames. We use learnable positional embeddings for each frame position and four transformer encoder layers [3]. Following the standard practice, we use the embedding at position 0 in the transformer's output as the final output embedding.

This architecture also allows for efficiency improvements. For example, during preprocessing, we can encode all video clips into embeddings and build an index for all the embeddings based on cosine similarity (e.g., using LSH [7]). Subsequent user queries are then quickly processed through this index. We considered an alternative model architecture (Figure 11) that uses a cross attention module [30] and an MLP classifier. However, our evaluation (§ 6.4) shows that the MLP classifier easily overfits the training data.

Let $e_{\text{anchor}}$ denote the encoded embedding of an anchor clip; let $e_{\text{pos}}$ and $e_{\text{neg}}$ denote the encoded embedding of the anchor's positive clip and negative clip generated via the simulator. We have the following intuitions for training:

- We would like to maximize the difference $d_{\text{sim}}$ between the similarity score of the anchor to the positive clip $cos(e_{\text{anchor}}, e_{\text{pos}})$ large and that of the anchor to the negative clip $cos(e_{\text{anchor}}, e_{\text{neg}})$.

- Consider two scenarios: (1) $cos(e_{\text{anchor}}, e_{\text{pos}}) = 0.99$ and $cos(e_{\text{anchor}}, e_{\text{neg}}) = 0.89$; (2) $cos(e_{\text{anchor}}, e_{\text{pos}}) = 0.39$ and $cos(e_{\text{anchor}}, e_{\text{neg}}) = 0.29$. In both cases, $d_{\text{sim}} = 0.1$. Intuitively, in scenario (1), we should make $cos(e_{\text{anchor}}, e_{\text{neg}})$ small as it is almost not possible to make $cos(e_{\text{anchor}}, e_{\text{pos}})$ larger. In contrast, in scenario (2), we should focus on making $cos(e_{\text{anchor}}, e_{\text{pos}})$ larger as the score is too small for a positive clip.

We use circle loss [50] as our objective function as it incorporates both of the above intuitions. During training, we generate random training data on the fly for each batch to maximize data diversity. We train the model until training loss converges.

**Data normalization.** We transform each 2D clip generated from the simulator to a suitable input format for the encoder model as follows. We first normalize the bounding box coordinates using their mean and standard deviation. Since it is easier for neural networks to use fixed-size input, we sub-sample or over-sample uniformly to ensure the clips $C_Q$ and $C_V$ have a fixed 128 frames, which is enough to represent a 1-minute video and is still understandable to humans. We limit our focus to at most four objects per clip, given that most queries usually involve only a few objects. In scenarios with fewer than four objects, we pad the sequence with zero-valued bounding boxes. Therefore, $C_Q = \{O_1, O_2, O_3, O_4\}$ includes four sequences of bounding boxes, where each sequence $O_i$ has a length of 128. We concatenate the four values of the bounding boxes for four objects in each frame and obtain a matrix $X_Q$ with size 128x16, which is used as input for the model. We obtain $X_V$ for $C_V$ similarly.

## 5 TUNER: INCORPORATING USER FEEDBACK

The trained encoder model serves as a cold start to query similar clips, and our experiments have shown that this already exceeds baseline performance (Section 6.2). However, given the ambiguous nature of visual content, there could be different user intents even with the same visual query. For example, consider the car making a left turn, then moving straight, and then a right turn query in Figure 4. The user might want the clips where the car eventually moves southward, or the user may not care about the direction at all. Therefore, we incorporate user feedback to adapt the pre-trained model to the user query at hand to better capture user intent by learning from the feedback.

The feedback-learning process works as follows. We first present users with clips retrieved by the default model sorted by similarity. The user then reviews these clips and labels some of them as either 'good' or 'bad' examples (e.g., labeling the 1st and the 3rd examples as positive and the 2nd and 4th examples as negative). The Tuner uses this feedback to adapt the pre-trained model and refine its understanding of the user's intent and preference. We can repeat this feedback loop until the user is satisfied with the results.

One challenge we face is the limited amount of labeled examples from user feedback. Therefore, we need the learning process to be effective while preventing overfitting. We develop four techniques to learn from limited user feedback effectively:

**Data Augmentation.** We augment the user-provided labeled examples by performing random rotation on the bounding box sequence and adding random white noise to each bounding box. We use data augmentation to generate three times the number of negative examples and then generate positive examples to ensure the number of positive pairs matches the number of negative pairs.

**Fine-tuning.** Let $SC_{\text{pos}}$ and $SC_{\text{neg}}$ denote the set of labeled positive examples and negative examples respectively after data augmentation. To simplify notation, we add the query $C_Q$ into $SC_{\text{pos}}$, *i.e.*, $C_Q \in SC_{\text{pos}}$, because the synthetically created query $C_Q$ is a positive example just like any other positive examples labeled by user. During finetuning, any pair of examples $C_i, C_j$ where $C_i \in SC_{\text{pos}}$ and $C_j \in SC_{\text{pos}}$ serve as a positive example for the model, and any pair of examples $C_i, C_j$ where $C_i \in SC_{\text{pos}}$ and $C_j \in SC_{\text{neg}}$ serve as a negative example for the model.

Intuitively, when there are more labeled examples, we trust the labeled examples more and fine-tune the model for more steps; when there are fewer labeled examples, we trust the pre-trained model more and fine-tune the model for fewer steps to avoid overfitting. Therefore, we set the number of finetuning steps as $\sqrt{|SC_{\text{pos}}| + |SC_{\text{neg}}|}$, proportional to the number of available labeled examples $|SC_{\text{pos}}| + |SC_{\text{neg}}|$.

**Layer-wise Decreasing Learning Rate.** A typical practice is to fine-tune only the last layer(s). This works well when the first layers of the pre-trained model can recognize almost all common low-level features for the task. For example, when adapting models pre-trained on ImageNet to downstream tasks, one only needs to finetune the last layers as the images in ImageNet include almost all possible low-level image features, and one only needs to adapt the high-level semantic features.

However, in our case, the model is pre-trained on synthetic data, which might lack some low-level variations in real data. Therefore, we propose fine-tuning all layers so that the model adapts low-level and high-level semantic features. Fine-tuning all layers can easily lead to overfitting. Inspired by existing work on fine-tuning large language models [49], we use layer-wise decreasing learning rates, assigning lower learning rates to the first and larger ones to the last layers. Specifically, we use learning rate $1e^{-5}$ (same as the learning rate used for pre-training) for the final layer and decay by a factor of 0.5 for each previous layer.

**Augmenting Query Set with Found Positive Examples.** Intuitively, any positive examples from the user feedback can serve as our query. Since these examples are from the actual video, using them as the query might increase recall. Therefore, we augment the query with user-labeled positive examples. Specifically, for the original sketched visual query, we obtain a similarity score $y_i$ for each video clip. Similarly, for each labeled positive example $C_j$, we obtain a similarity score $y_i^j$ for each clip. Let's say there are $N_{\text{pos}}$ positive examples. We obtain a final similarity score by incorporating all positive examples as $y_i^* = \frac{y_i + \sum_j y_i^j}{1 + N_{\text{pos}}}$. We then use $y_i^*$ as the prediction for each video clip and rank the clips based on $y_i^*$.

During the user-feedback loop, the number of found positive examples $N_{\text{pos}}$ increases. Since we need to use each positive example as the query for prediction, the time complexity increases. However, this can be easily parallelized. In addition, we expect the user to only label a few examples (and even fewer positive examples), so the increased time complexity is computationally tractable.

# 6 EXPERIMENTS

In our experiments, we demonstrate the effectiveness of SketchQL by investigating the following questions:

- How does SketchQL compare against state-of-the-art approaches in accurately retrieving relevant video moments? (§6.2)
- How does SketchQL's visual query interface compare to natural language- and SQL-like interfaces in its ease of use? (§6.3)
- How much do each of SketchQL's components contribute to the final accuracy? (§6.4).
- What is the runtime performance of SketchQL? (§6.5)

## 6.1 Experimental setup

**Datasets.** We use the following datasets:

- VIRAT [42]. This is a traffic surveillance video benchmark dataset used extensively in the computer vision community. We use the longest video from the dataset with 7k frames.
- BDD100k [58]. This is a driving video dataset of dashcam footage. We use a subset of 3.5k frames from the dataset.
- SoccerNet [24]. This is a dataset of broadcast soccer games and is a popular benchmark dataset. We use the labeled subset (with ground-truth object tracking results) totaling 43k frames.
- YouTube-8M [6]. This is a public video classification dataset with YouTube videos from diverse domains. We use a Broadcast American college football video containing 11k frames.

Three datasets, VIRAT, BDD100k, and YouTube-8M, do not have ground-truth object bounding box trajectories, so we use Bytetrack [64] object tracker to obtain the object bounding box trajectories. The SoccerNet dataset has ground-truth object bounding box trajectories, and each person object has the annotation "player_left" for players in the left team and "player_right" for players in the right team. Notably, our method *d*oes not access the datasets during pre-training; datasets are only used during evaluation and to fine-tune our model in the user feedback learning experiment in §6.2.3. The traffic datasets VIRAT and BDD100k have stable cameras in most frames, while the sports broadcast datasets SoccerNet and YouTube-8M contain camera movement and are more challenging.

**Baselines.** We evaluate SketchQL against these baselines:

- NL-Clip. This method uses a natural language interface. Similar to Zelda [46], it is adapted from the large vision-language model Clip from OpenAI [43]. It is trained on a large corpus of image-text pairs. Given a text query, it retrieves relevant frames from the video.
- NL-2DTAN [62, 63]. This is a popular natural language-based method for video moment retrieval that is trained directly on video clips and associated captions. Since we consider the zero training data scenario, we use a pre-trained version of 2DTAN that was trained on ActivityNet [10]. *We observed that 2DTAN has zero retrieval accuracy on all the queries in our datasets since the videos differ from the training set, so we omit this baseline from our results.*
- SQL-Track. This method uses a SQL-like interface where users can retrieve video clips by writing rules over object trajectories. Notably, similar to SketchQL, it uses object trajectories as the primitive. This baseline is similar to Miris [9]. Both methods return the same output clips, while Miris additionally accelerates the query execution.
- SQL-Scene. This method also uses a SQL-like interface where users retrieve video clips by writing rules over scene graphs [33]. We use the state-of-the-art model [56] to extract scene graphs for this method. This baseline is similar to EQUI-VOCAL [60] on scene graphs. The main difference is that, unlike EQUI-VOCAL, where queries are synthesized automatically, SQL-Scene queries are constructed by users with best efforts over the scene graphs.

| Dataset | Query ID | Query |
|---|---|---|
| VIRAT | Q1.1 | A car makes a left turn [9]. |
| | Q1.2 | A car makes a left turn and then a right turn. |
| | Q1.3 | A car stops to yield to a pedestrian that crosses the road. |
| BDD100k | Q2.1 | A pedestrian crosses the street on a crosswalk [9, 46]. |
| SoccerNet | Q3.1 | A player kicks the ball [20]. |
| | Q3.2 | A player kicks the ball into the air. |
| | Q3.3 | A player passes the ball to one of his teammates but an opponent player tries to intercept the ball. |
| YouTube-8M | Q4.1 | An opponent tackles a running player in an American football match. |
| | Q4.2 | A player dashes forward with two other players running behind. |

**Table 2: Queries for each dataset.**

For a fair comparison, the main results are reported without user feedback learning. We evaluate the impact of user feedback learning on SketchQL in §6.2.3.

**Setup for all methods.** For SketchQL, we pre-train on synthetic data using the Adam optimizer at a learning rate $1e^{-5}$ and a batch size of 500. We generate training data on the fly for each batch to ensure maximum training data diversity. We train the model until convergence, which takes about 7 days on an A40 GPU.

We use the pre-trained ViT-B/32 model for NL-Clip [1] and the pre-trained PSGTR model with a ResNet-50 backbone for SQL-Scene [2]. Since NL-Clip and SQL-Scene operate on individual frames, we first retrieve the top-K similar frames and generate video clips using a fixed number of frames before and after the matching frame. For SQL-Track, we implement handcrafted rules for each query in Python (similar to Listing 1) with up to 100 minutes per query for rule creation and manual parameter tuning to optimize performance. Rules are reused where possible across queries.

**Queries.** Table 2 presents the queries evaluated for each dataset, where citations denote that the same queries were used in the cited papers. Not all queries used in prior works are supported by our visual interfaces (discussed in § 3), and we have included new queries that emphasize the movement patterns of objects. Each query is created using the interface supported by its respective method: natural language for NL-Clip, a SQL-like language (*i.e.*, rules) for SQL-Track and SQL-Scene, and visual queries for SketchQL. Table 3 shows how query Q3.1 is represented in each of the baselines. We also include samples of found clips by SketchQL for the queries in our artifacts [4].

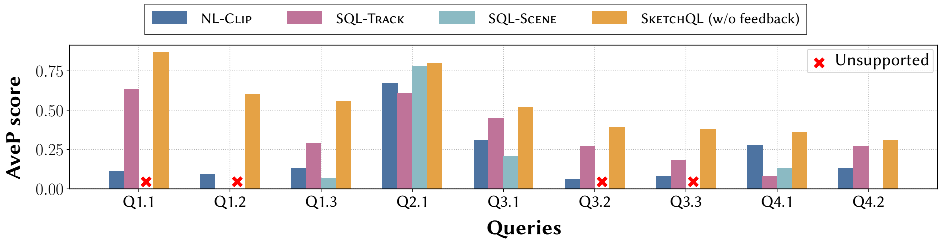


**Figure 8: AveP score comparison of SketchQL (without user feedback) against other baselines**

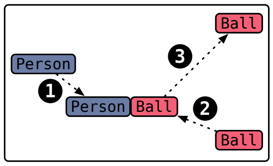
| Method | Query format |
|---|---|
| NL-Clip | "A player kicks a ball on a soccer field" |
| SQL-Track | `SELECT person, ball FROM ObjectTrackUDF(video)`<br>`WHERE MinDistance(person, ball)=0`<br>`AND RelativeSpeed(person, ball)>0`<br>`AND BBoxY(ball) >= 0.75*BBoxY(person)`<br>`AND BBoxY(ball) < BBoxY(person)` |
| SQL-Scene | `SELECT frame, graph FROM SceneGraphUDF(video)`<br>`WHERE graph.subject="person"`<br>`AND graph.predicate="kicking"`<br>`AND graph.object="ball"` |
| SketchQL |  |

**Table 3: Query specification in SketchQL and other systems.**

**Performance Metric.** Performance metrics like precision and recall do not account for clip ranking. Since we favor true positives ranked higher, we adopt the AveP score, commonly used in information retrieval, as our performance metric [36, 45]. Using the top-k retrieved clips, precision and recall are calculated for each $k$. As $k$ increases, the recall improves. Let $p(r)$ denote the precision at recall $r$. AveP is defined as the area under the precision-recall curve [16]: AveP = $\int_{r=0}^{1} p(r)dr$. Intuitively, AveP is higher when more true positives are ranked higher, with the maximal AveP=1 when all $n_{\text{pos}}$ positive instances are ranked in top $n_{\text{pos}}$. A clip is considered to be a positive match if its overlap ratio (size of intersection/size of the union) with a ground-truth clip is more than 50%. The ground-truth clips are obtained by our manual inspection of each video in all datasets. For each query, we consider top 10×$n_{pos}$ clips from each method to numerically compute the AveP score where $n_{pos}$ is the number of ground-truth positive examples.

**Hardware Setup.** We run the experiments on a server with one NVIDIA A40 RTX GPU, 120 Intel CPU cores, and 384 GB of RAM. We use the GPU for object tracking, model training, and inference.

## 6.2 End-to-end Query Performance

*6.2.1 Quantitative Results.* We execute the query for each system and evaluate the quality (measured by AveP) of the retrieved results. The results are shown in Figure 8.

The natural language-based method NL-Clip has a low AveP score on six of the nine queries. This is because natural language-based methods require training on a large amount of labeled data for each dataset to perform well. Additionally, NL-Clip only supports frame-level querying. This results in its satisfactory performance on Q2.1 and poor performance on temporal queries such as Q1.3. We tested a different natural language method, 2DTAN, for video-level querying. However, we observed that 2DTAN achieves a 0 AveP score on all the queries. In our "zero-shot" scenario with no training data for each dataset, pre-trained natural language-based methods do not generalize well due to data distribution differences from their original training set. Finally, we observe that NL-Clip performs better when the query can be identified using *keywords*. For example, Q4.1 is more heavily dependent on the keyword "tackle", which allows NL-Clip to retrieve some relevant clips.

SQL-Track is the best-performing baseline method. SQL-Track operates on object trajectory primitives that can be obtained more reliably across datasets. As long as the user writes good UDFs (*i.e.*, rules), it can achieve good performance. At the same time, its performance is limited due to the reliance on the user's expertise to write good UDFs. For example, for Q1.1, the rules written by a graduate student got a 0 AveP score. The student had to debug and tune the parameters multiple times to achieve the reported performance, which was a time-consuming and tedious process.

SQL-Scene does not support four of the nine queries. While Q1.1 and Q1.2 are single-object queries (involving only one car) that cannot be represented by scene graphs, Q3.2 and Q3.3 involve complex actions (*e.g.*, kicking the ball *into the air*) that are unavailable in the scene graph taxonomy. SketchQL outperforms SQL-Scene in all the supported queries. We observe that the pre-trained scene-graph models do not generalize well and perform poorly on difficult sports broadcast datasets with more camera movements.

Overall, SketchQL outperforms all the baselines by 20% on average. This is because (1) similar to SQL-Track, SketchQL leverages object trajectories that can be obtained more reliably across datasets, and (2) unlike SQL-Track, SketchQL does not rely on the user's expertise to write good UDFs; users can easily create the queries by simple drag-and-drop gestures.

**Limitations.** SketchQL's performance declines in four scenarios:
(1) Object tracking failures. State-of-the-art object trackers like Bytetrack [64] do not perform well when there is occlusion, causing tracking failures (*e.g.*, trees blocking cars in VIRAT [42]).
(2) Unstable camera. Camera movement adversely affects the relative trajectories of objects. For instance, a windy environment in the VIRAT dataset leads to camera shakes. This results in SketchQL retrieving some cars moving straight but appearing to turn left.
(3) Loss of semantic information. SketchQL relies on bounding boxes, so it loses some semantic information. Nevertheless, SketchQL can be used to retrieve the "syntactically" correct clips, and then a more expensive semantic model can be used for further filtering.

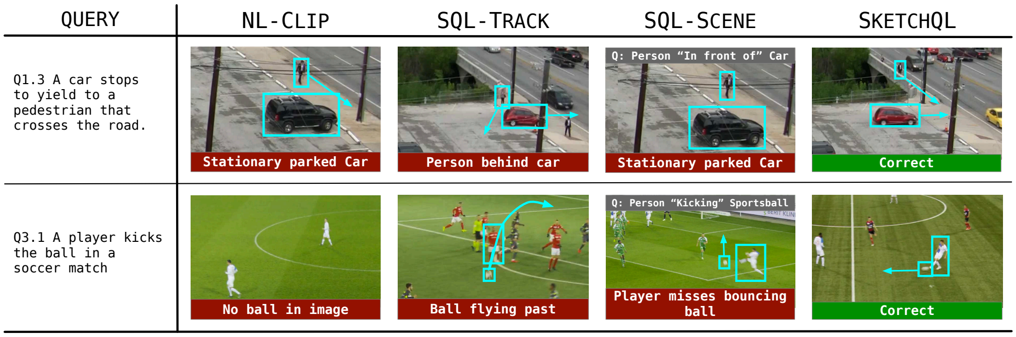

QUERY
NL-CLIP
SQL-TRACK
SQL-SCENE
SKETCHQL
Q1.3 A car stops to yield to a pedestrian that crosses the road.
Stationary parked Car
Person behind car
Q: Person "In front of" Car
Stationary parked Car
Correct
Q3.1 A player kicks the ball in a soccer match
No ball in image
Ball flying past
Q: Person "Kicking" Sportsball
Player misses bouncing ball
Correct

**Figure 9: Qualitative comparison of SketchQL against other baselines. Details of the failures discussed in §6.2.2.**

| | Q1.1 | Q1.2 | Q1.3 | Q2.1 | Q3.1 | Q3.2 | Q3.3 | Q4.1 | Q4.2 |
|---|---|---|---|---|---|---|---|---|---|
| w/o Feedback | 0.87 | 0.60 | 0.56 | 0.80 | 0.52 | 0.39 | 0.38 | 0.36 | 0.31 |
| w/ Feedback | **0.93** | **0.73** | 0.56 | 0.80 | **0.58** | **0.44** | **0.58** | 0.36 | 0.31 |

**Table 4: Performance of SketchQL without and with user feedback.**

(4) User intent ambiguity. Users may sketch the same visual query for different intents. To address this, SketchQL supports a post-processing step where users can use rules like SQL-Track to filter out clips. Since writing rules is difficult, SketchQL can also refine results for each query by learning from user feedback (*i.e.*, labels) on retrieved clips (§6.2.3).

*6.2.2 Qualitative Results.* The qualitative results for different systems are shown in Figure 9. Both NL-Clip and SQL-Scene works best when processing individual frames, so they fail to recognize clips with moving cars in Q1.3. In contrast, SketchQL and SQL-Track use trajectory-based matching to accurately detect object movement. Additionally, NL-Clip's pre-trained model struggles to detect the ball in Q3.1. The object tracking primitives used by SketchQL and SQL-Track better handle such difficult detections. SQL-Track heavily relies on the user's expertise to write proximity and displacement rules to identify individual object movements. However, these rules written within a limited time span are prone to errors. For example, in Q3.1, we observe that SQL-Track incorrectly retrieves clips that contain the ball flying past a player since they match all the rules for *player kicks ball.* In contrast, SketchQL does not rely on the user's expertise and can accurately match the visual queries to the video moments using the learned model, making it less prone to errors.

*6.2.3 Query Performance with User Feedback.* We evaluate the performance of SketchQL with human feedback. Since the baseline methods do not support learning from user feedback, we only show results for our method. We initially run SketchQL with the visual query and label the top-5 retrieved clips. We then incorporate this feedback using the process described in §5 to refine the model. Finally, we retrieve clips with the refined model and report the AveP score on the newly retrieved clips.

The results are shown in Table 4. Overall, by incorporating user feedback, the performance is improved in 5 out of 9 queries by 5.5% on average. However, incorporating the feedback does not change the performance in a few cases: (1) The top 5 clips are all correct (*e.g.*, for Q2.1), so the feedback does not provide additional information. (2) The performance is limited by the noise in the video (*e.g.*, Q4.1 and Q4.2), including camera movements and object tracking errors.

## 6.3 Ease-of-use Evaluation

*6.3.1 User Effort.* We investigate the amount of user effort, measured in time, required to express the queries using different query interfaces. Table 5 shows the time taken for a graduate student to compose each query via each interface. This time includes the duration from the start of the query composition to the point of achieving a fully functioning query.

| | Q1.1 | Q1.2 | Q1.3 | Q2.1 | Q3.1 | Q3.2 | Q3.3 | Q4.1 | Q4.2 |
|---|---|---|---|---|---|---|---|---|---|
| NL-Clip | <1 | <1 | <1 | <1 | <1 | <1 | <1 | <1 | <1 |
| SQL-Track (from scratch) | 90 | 100 | 20 | 40 | 10 | 90 | 40 | 100 | 45 |
| SQL-Track (reuse rules) | 90 | 10 | 20 | 30 | 10 | 30 | 20 | 80 | 30 |
| SQL-Scene | - | - | <1 | <1 | <1 | - | - | 2 | <1 |
| SketchQL | <1 | <1 | 2 | <1 | 2 | 2 | 4 | 4 | 3 |

**Table 5: Query composition time (minutes) for different interfaces.**

NL-Clip requires the least time as the user can trivially express each query in natural language. For SQL-Track, we report two variants: (1) From scratch - where we write the required rules (UDFs) from scratch for each query (2) Reuse rules - where we develop queries one at a time, and for each query, we reuse existing rules from previous queries where possible. In both cases, SQL-Track requires significantly more time than other methods since it is non-trivial to express notions like "left-turn" with SQL-like rules.

SQL-Scene requires low effort to represent the query, as the user can define the query as a relationship between objects detected in the scene graphs. For example, query Q2.1 can be represented in the scene-graph model as the (subject, relationship, object) tuple of ('person', 'kicking', 'ball'). However, scene graph models only support simple relationships between the subject and object, such as 'kicking', 'in front of', 'on' etc. So, non-trivial events such as the orientation-agnostic left turn of a car or a player kicking the ball into the air cannot be retrieved using this baseline.

In contrast, users can efficiently compose all queries in SketchQL by simply dragging and dropping the objects with the mouse in any direction. Generally, SketchQL requires more time when more

|  |  | + Rotation degrees |  |  | + Noise level |  |  |
|---|---|---|---|---|---|---|---|
|  | Q1.1 | 90 | 180 | 270 | 5% | 10% | 15% |
| AveP score | 0.87 | 0.86 | 0.87 | 0.87 | 0.87 | 0.85 | 0.82 |

**Table 6: Performance of SketchQL on Q1.1 under 1) different degrees of rotation and 2) different levels of white noises.**

objects are involved in the query as one needs to create each object individually and align their motions using the Trajectory Panel.

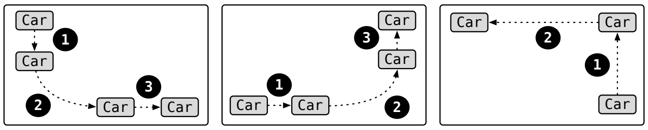


**Figure 10: Variants of user sketches for the left turn query.**

*6.3.2 Query Sensitivity.* This experiment measures the sensitivity of the system to user sketches. For example, Figure 10 shows different ways a user can depict a left turn query. We observe that SketchQL produces similar AveP scores for all the variants without any fine-tuning. This can be attributed to SketchQL's synthetic training data generation process that automatically generates a diverse set of trajectories, described in §4.2. In contrast, baseline methods like SQL-Track require writing new rules to support each variant.

For a more quantitative assessment, we consider two variations: rotation and white noise.

**Rotation.** Table 6 shows the AveP scores under query rotations of 90, 180, and 270. The scores remain stable under different degrees of rotation. We observe the rankings of the retrieved clips are almost identical, with only minor variations like the 5th and 6th clips swapping places. Thus, SketchQL is robust to different query rotations. The robustness stems from our training data generation method that considers rotation variation.

**White Noise.** We add white noise to the query to simulate wobbly trajectories in the query that humans may create when drawing queries by hand/mouse. We set the magnitude of the white noise to 5%, 10%, and 15% of the scale of the query trajectory (measured by the standard deviation of coordinates). The results are shown in Table 6. When the noise level increases, the AveP score decreases. However, the system remains robust, maintaining performance for noise up to 10% of the overall query trajectory scale.

To summarize, SketchQL is robust to different user sketch variations, so query specification is easy.

## 6.4 Alternative Matcher Designs

We evaluate alternative designs of the Matcher component discussed in §4.3. Specifically, we consider (1) replacing the learned model with classic time series similarity search using manually extracted features and (2) using a different model architecture.

**Time series similarity search with manual features.** This is the baseline we discussed in § 4.1 where we convert the problem to a time series similarity search under Euclidean or DTW distance [57]. We manually extract diverse features using domain knowledge. The features include (1) absolute positions, *i.e.*, trajectories, (2) velocities, (3) angles, and (4) relational features (e.g., distances) between object pairs. We also normalize each feature to ensure all features are at the same scale. We use Euclidean distance (DTW distance gives similar results) to compute the similarity between the query and video clip features. We also manually tune the weights for each feature dimension to give this baseline an advantage.

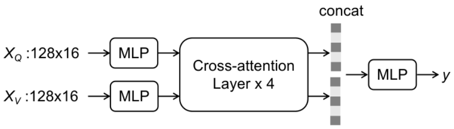


**Figure 11: Alternative Model Architecture**

**Alternative model architecture.** When encoding clip $C_V$, one intuition is to be aware of the query $C_Q$. For example, if the query trajectory is circular, the model could focus more on trajectory features in $C_V$ to match that pattern. This motivates the model architecture shown in Figure 11, which features a cross-attention module [30] for the clip and query to pay attention to each other.

|  | Q1.1 | Q1.2 | Q1.3 | Q2.1 | Q3.1 | Q3.2 | Q3.3 | Q4.1 | Q4.2 |
|---|---|---|---|---|---|---|---|---|---|
| SketchQL | **0.87** | **0.60** | **0.56** | **0.80** | **0.52** | **0.39** | **0.38** | **0.36** | **0.31** |
| *manual feat | 0.24 | 0.0 | 0.12 | 0.49 | 0.21 | 0.17 | 0.08 | 0.13 | 0.15 |
| *alt arch | 0.28 | 0.17 | 0.56 | 0.61 | 0.42 | 0.21 | 0.27 | 0.28 | 0.24 |

**Table 7: Ablation Study: AveP scores.**

As shown in Table 7, replacing any component substantially decreases performance, especially when using manual features with classic distance functions instead of the pre-trained model. This is because (1) metrics like Euclidean distance and dynamic time warping are not invariant to different camera angles (Figure 6), while the learned model ensures invariance; (2) The manual features are limited by human knowledge of crafting the features while the learned model can consider subtle features that are less obvious to humans; (3) It is very difficult to manually balance the importance of different features. The alternative architecture performs poorly because it has a built-in MLP classifier, which can easily overfit the training data. In contrast, our proposed architecture focuses on learning feature representations and is thus less prone to overfitting than an MLP classifier.

## 6.5 Efficiency analysis

We report the running time of different methods on the queries.

**Primitives Computation.** Table 8 reports the time for primitive computation for each method. For SoccerNet, the object tracks are provided in the dataset, so we omit this step. SQL-Track and SketchQL both use the same object tracker, Bytetrack [64], so their running time is identical. SQL-Scene requires more expensive scene graph computation. Additionally, in three of the four datasets, SQL-Scene requires computing scene graphs in a 2 × 2 grid to achieve reasonable accuracy, increasing the time 3×. For NL-Clip, we precompute and save the image embeddings for each frame. For all methods, primitive computation is a one-time process, and the computed primitives can be reused for all future queries.

**Query Execution.** SketchQL's execution time is generally on par with other baselines. The runtime for SketchQL, however, may fluctuate when there are more objects in the query. It may increase

| | VIRAT | BDD100K | SoccerNet | Youtube-8M |
|---|---|---|---|---|
| NL-Clip | 77 | 13 | 458 | 76 |
| SQL-Track | 335 | 151 | - | 417 |
| SQL-Scene | 1139 | 152 | 5728 | 1189 |
| SketchQL | 335 | 151 | - | 417 |

**Table 8: Time (seconds) required for primitive computation. SQL-Track and SketchQL use object tracking while SQL-Scene uses scene graphs. NL-Clip uses image embeddings for each frame.**

(Q3.3) when the number of possible combinations for finding the one-to-one object mapping increases. It may decrease (Q1.3) when the additional objects do not appear often and serve as early filters.

| | Q1.1 | Q1.2 | Q1.3 | Q2.1 | Q3.1 | Q3.2 | Q3.3 | Q4.1 | Q4.2 |
|---|---|---|---|---|---|---|---|---|---|
| NL-Clip | 28 | 29 | 29 | 2 | 136 | 142 | 152 | 36 | 34 |
| SQL-Track | 70 | 75 | 110 | 3 | 53 | 57 | 64 | 6 | 24 |
| SQL-Scene | - | - | 1 | 1 | 2 | - | - | 2 | 2 |
| SketchQL | 12 | 14 | 8 | 1 | 19 | 20 | 67 | 6 | 14 |

**Table 9: Running time (seconds).**

**Model Refinement.** When users provide feedback, SketchQL refines the model and re-executes the query. The time required for model refinement is roughly the same for all queries and is about 2.9 seconds. The time is short because we are only finetuning the model for a few steps with a few examples.

## 7 RELATED WORK

**Query Interfaces.** SoTA VDBMSs support two interfaces for querying video moments: natural language-based interface [11, 22, 38, 53, 61] and SQL-based interface [9, 12, 13, 21, 34, 37, 52, 54]. Natural language-based methods require a large amount of training data and do not generalize well on datasets different from the training data. SQL-like methods are difficult to use, requiring a considerable amount of query specification time. There is also existing work to simplify query specification for SQL-like queries through automatic query synthesis [60]. For example, EQUI-VOCAL [60] automatically synthesizes SQL-like queries from a few labeled examples. However, the user would need to find the few labeled examples first, which can require non-trivial human effort if the event of interest is rare. Furthermore, EQUI-VOCAL uses scene graphs as the data model, so its performance depends on the quality of scene graphs, which currently can not be extracted as reliably as object trajectories. In addition, single object queries like *car left turn* are unsupported under this data model, since scene graphs must involve at least two objects.

Our visual query language (Sketcher) provides an intuitive visual interface with high usability, and our query execution component (Matcher) provides high generalizability across datasets.

**Visual query language for videos.** There were early attempts to develop visual query languages for videos in the 2000s [18, 28, 29, 35]. These methods provide GUI components for some predefined predicates, and users can use them to compose queries in a visual interface. Example predicates include temporal predicates like "start at" and "finish by" [28, 29] and spatial predicates like "overlap" and "cover" [35]. These visual query languages are more like a visual representation of SQL-like languages and are limited by predefined predicates. In addition, they suffer from the same limitation as SQL-like languages in that the user has to translate what he wants to be rules composed by the predefined predicates (in the form of GUI or not), which can be non-trivial. In contrast, in our proposed visual query language, we obliviate the need for predefined predicates, and users can freely draw animated sketches of the target event. The visual query is an animation of the target event, so what the user sees in the visual query is what he/she wants to query, which is more intuitive and expressive.

**Video data management systems.** A line of existing video data management systems focuses on improving the efficiency of video analytics queries. For machine learning inference queries, PP [41] and BlazeIt [34] use lightweight models to filter out irrelevant frames, and EVA [54] reuses results across queries. For object track queries, MIRIS [9] processes the video at low framerates when possible. Our work SketchQL focuses more on the query interface and features a visual query language for video moment retrieval.

**Spatiotemporal information-based video retrieval.** There are also existing methods that use spatiotemporal information for video retrieval. For example, methods use spatiotemporal descriptors as features to train a model [17] and trajectory shape-based matching using manually designed similarity functions [31, 32]. A recent work STAR-Retrieval [12] formalizes the retrieval problem as a graph-matching problem and supports fuzzy matching through discretization (e.g., discretizing distance between two objects). However, STAR-Retrieval is sensitive to the orientation of the objects in the query due to its definition of angles between two objects. User-provided sketch of a single object trajectory is also explored as the query interface [31, 32]. However, in existing methods, the sketch is drawn on the video frame, and the absolution location information is used for matching (the goal is to find "near exact" matches). For example, on a traffic surveillance video, the sketch may be on one particular road lane, and existing methods aim to find objects that follow the trajectory in that lane. Therefore, these sketches are tied to a particular video, and even in that video, the retrieved trajectories are tied to a particular location, orientation, and scale. STAR-Retrieval [12] supposedly supports user-provided sketch in the form of a sequence of graphs, but no detail on the sketch interface is available. The sketch represents spatial relationships as distance and angle parameters, similar to predicates in SQL. SketchQL supports multiple objects and is designed to be dataset/location/orientation/scale independent.

## 8 CONCLUSION

Video moment querying is an important yet challenging task for video analytics. Existing techniques for this task suffer from poor ease of use and generalizability. In this paper, we presented SketchQL, a novel visual querying system that enables intuitive video moment retrieval through sketch-based queries. SketchQL allows users to visually depict queries using the Sketcher interface. It then matches user queries to video moments using a transformer model-based Matcher trained on diverse synthetic data. Furthermore, SketchQL incorporates user feedback via the Tuner to improve retrieval accuracy. Our experiments demonstrate that SketchQL significantly improves the usability and retrieval accuracy over state-of-the-art methods.

# REFERENCES

[1] 2021. Openai/CLIP: CLIP (Contrastive Language-image pretraining), predict the most relevant text snippet given an image. https://github.com/openai/CLIP [Online; accessed 30. Aug. 2023].

[2] 2022. Benchmarking Panoptic Scene Graph Generation (PSG), ECCV'22. https://github.com/Jingkang50/OpenPSG [Online; accessed 30. Aug. 2023].

[3] 2023. TransformerEncoder — PyTorch 2.0 documentation. https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoder.html [Online; accessed 8. Apr. 2023].

[4] 2024. code and sample clips. https://figshare.com/s/da2add67051616fbf5de

[5] 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.

[6] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Apostol (Paul) Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. 2016. YouTube-8M: A Large-Scale Video Classification Benchmark. In *arXiv:1609.08675*. https://arxiv.org/pdf/1609.08675v1.pdf

[7] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. 2015. Practical and optimal LSH for angular distance. *Advances in neural information processing systems* 28 (2015).

[8] 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.

[9] 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.

[10] Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. 2015. Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the ieee conference on computer vision and pattern recognition*. 961–970.

[11] Long Chen, Chujie Lu, Siliang Tang, Jun Xiao, Dong Zhang, Chilie Tan, and Xiaolin Li. 2020. Rethinking the bottom-up framework for query-based video localization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 10551–10558.

[12] 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.

[13] Yueting Chen, Xiaohui Yu, Nick Koudas, and Ziqiang Yu. 2021. Evaluating temporal queries over video feeds. In *Proceedings of the 2021 International Conference on Management of Data*. 287–299.

[14] Pramod Chunduri, Jaeho Bang, Yao Lu, and Joy Arulraj. 2022. Zeus: Efficiently localizing actions in videos using reinforcement learning. In *Proceedings of the 2022 International Conference on Management of Data*. 545–558.

[15] Contributors to Wikimedia projects. 2022. Pinhole camera model - Wikipedia. https://en.wikipedia.org/w/index.php?title=Pinhole_camera_model&oldid=1110164612 [Online; accessed 9. Apr. 2023].

[16] Contributors to Wikimedia projects. 2023. Evaluation measures (information retrieval) - Wikipedia. https://en.wikipedia.org/w/index.php?title=Evaluation_measures_(information_retrieval)&oldid=1138907815 [Online; accessed 19. Mar. 2023].

[17] Daniel DeMenthon and David Doermann. 2003. Video retrieval using spatio-temporal descriptors. In *Proceedings of the eleventh ACM international conference on Multimedia*. 508–517.

[18] Mehmet Emin Dönderler, Ediz Şaykol, Umut Arslan, Özgür Ulusoy, and Uğur Güdükbay. 2005. BilVideo: Design and implementation of a video database management system. *Multimedia Tools and Applications* 27, 1 (2005), 79–104.

[19] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An open urban driving simulator. In *Conference on robot learning*. PMLR, 1–16.

[20] Na Feng, Zikai Song, Junqing Yu, Yi-Ping Phoebe Chen, Yizhu Zhao, Yunfeng He, and Tao Guan. 2020. SSET: a dataset for shot segmentation, event detection, player tracking in soccer videos. *Multimedia Tools and Applications* 79 (2020), 28971–28992.

[21] Daniel Y Fu, Will Crichton, James Hong, Xinwei Yao, Haotian Zhang, Anh Truong, Avanika Narayan, Maneesh Agrawala, Christopher Ré, and Kayvon Fatahalian. 2019. Rekall: Specifying video events using compositions of spatiotemporal labels. *arXiv preprint arXiv:1910.02993* (2019).

[22] Jiyang Gao, Chen Sun, Zhenheng Yang, and Ram Nevatia. 2017. Tall: Temporal activity localization via language query. In *Proceedings of the IEEE international conference on computer vision*. 5267–5275.

[23] Junyu Gao and Changsheng Xu. 2021. Fast video moment retrieval. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1523–1532.

[24] Silvio Giancola, Mohieddine Amine, Tarek Dghaily, and Bernard Ghanem. 2018. Soccernet: A scalable dataset for action spotting in soccer videos. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 1711–1721.

[25] Joachim Gudmundsson and Michael Horton. 2016. Spatio-temporal analysis of team sports–a survey. *arXiv preprint arXiv:1602.06994* (2016).

[26] Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. 2015. Activitynet: A large-scale video benchmark for human activity understanding. In *2015 IEEE conference on computer vision and pattern recognition (CVPR)*. IEEE, 961–970.

[27] Fabian Caba Heilbron and Juan Carlos Niebles. 2014. Collecting and Annotating Human Activities in Web Videos. In *Proceedings of International Conference on Multimedia Retrieval*. ACM, 377.

[28] Stacie Hibino and Elke A Rundensteiner. 1995. A visual query language for identifying temporal trends in video data. In *Proceedings. International Workshop on Multi-Media Database Management Systems*. IEEE, 74–81.

[29] Stacie Hibino and Elke A Rundensteiner. 1997. MMVIS: Design and implementation of a multimedia visual information seeking environment. In *Proceedings of the fourth ACM international conference on Multimedia*. 75–86.

[30] Ruibing Hou, Hong Chang, Bingpeng Ma, Shiguang Shan, and Xilin Chen. 2019. Cross attention network for few-shot classification. *Advances in neural information processing systems* 32 (2019).

[31] Weiming Hu, Dan Xie, Zhouyu Fu, Wenrong Zeng, and Steve Maybank. 2007. Semantic-based surveillance video retrieval. *IEEE Transactions on image processing* 16, 4 (2007), 1168–1181.

[32] Yonggang Jin and Farzin Mokhtarian. 2004. Efficient Video Retrieval by Motion Trajectory.. In *BMVC*. Citeseer, 1–10.

[33] Justin Johnson, Ranjay Krishna, Michael Stark, Li-Jia Li, David Shamma, Michael Bernstein, and Li Fei-Fei. 2015. Image retrieval using scene graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3668–3678.

[34] 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).

[35] Sudhir Kaushik and Elke A Rundensteiner. 1998. SVIQUEL: A spatial visual query and exploration language. In *DEXA*. 290–299.

[36] Kazuaki Kishida. 2005. *Property of average precision and its generalization: An examination of evaluation indicator for information retrieval experiments*. National Institute of Informatics Tokyo, Japan.

[37] Nick Koudas, Raymond Li, and Ioannis Xarchakos. 2020. Video monitoring queries. *IEEE Transactions on Knowledge and Data Engineering* 34, 10 (2020), 5023–5036.

[38] 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.

[39] Meng Liu, Xiang Wang, Liqiang Nie, Xiangnan He, Baoquan Chen, and Tat-Seng Chua. 2018. Attentive moment retrieval in videos. In *The 41st international ACM SIGIR conference on research & development in information retrieval*. 15–24.

[40] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. 2018. Accelerating machine learning inference with probabilistic predicates. In *SIGMOD*. 1493–1508.

[41] 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.

[42] 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.

[43] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*. PMLR, 8748–8763.

[44] Michaela Regneri, Marcus Rohrbach, Dominikus Wetzel, Stefan Thater, Bernt Schiele, and Manfred Pinkal. 2013. Grounding action descriptions in videos. *Transactions of the Association for Computational Linguistics* 1 (2013), 25–36.

[45] Stephen Robertson. 2008. A new interpretation of average precision. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. 689–690.

[46] Francisco Romero, Caleb Winston, Johann Hauswald, Matei Zaharia, and Christos Kozyrakis. 2023. Zelda: Video Analytics using Vision-Language Models. arXiv:2305.03785 [cs.DB]

[47] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, et al. 2020. Lgsvl simulator: A high fidelity simulator for autonomous driving. In *2020 IEEE 23rd International conference on intelligent transportation systems (ITSC)*. IEEE, 1–6.

[48] 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.

[49] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune bert for text classification?. In *Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, October 18–20, 2019, Proceedings 18*. Springer, 194–206.

[50] Yifan Sun, Changmao Cheng, Yuhan Zhang, Chi Zhang, Liang Zheng, Zhongdao Wang, and Yichen Wei. 2020. Circle loss: A unified perspective of pair similarity optimization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 6398–6407.

[51] Stefanie Tellex and Deb Roy. 2009. Towards surveillance video search by natural language query. In *Proceedings of the ACM International Conference on Image and Video Retrieval*. 1–8.

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

[53] Shaoning Xiao, Long Chen, Songyang Zhang, Wei Ji, Jian Shao, Lu Ye, and Jun Xiao. 2021. Boundary proposal network for two-stage natural language video localization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 2986–2994.

[54] 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.

[55] Jian-Ru Xue, Jian-Wu Fang, and Pu Zhang. 2018. A survey of scene understanding by event reasoning in autonomous driving. *International Journal of Automation and Computing* 15, 3 (2018), 249–266.

[56] Jingkang Yang, Yi Zhe Ang, Zujin Guo, Kaiyang Zhou, Wayne Zhang, and Ziwei Liu. 2022. Panoptic Scene Graph Generation. In *ECCV*.

[57] Byoung-Kee Yi, Hosagrahar V Jagadish, and Christos Faloutsos. 1998. Efficient retrieval of similar time sequences under time warping. In *Proceedings 14th International Conference on Data Engineering*. IEEE, 201–208.

[58] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. 2020. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2636–2645.

[59] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. 2020. A survey of autonomous driving: Common practices and emerging technologies. *IEEE access* 8 (2020), 58443–58469.

[60] Enhao Zhang, Maureen Daum, Dong He, Magdalena Balazinska, Brandon Haynes, and Ranjay Krishna. 2023. EQUI-VOCAL: Synthesizing Queries for Compositional Video Events from Limited User Interactions [Technical Report]. *arXiv preprint arXiv:2301.00929* (2023).

[61] Hao Zhang, Aixin Sun, Wei Jing, and Joey Tianyi Zhou. 2020. Span-based localizing network for natural language video localization. *arXiv preprint arXiv:2004.13931* (2020).

[62] Songyang Zhang, Houwen Peng, Jianlong Fu, Yijuan Lu, and Jiebo Luo. 2021. Multi-scale 2d temporal adjacency networks for moment localization with natural language. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 12 (2021), 9073–9087.

[63] 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.

[64] 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 *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXII*. Springer, 1–21.

[65] Qihuang Zhong, Liang Ding, Li Shen, Peng Mi, Juhua Liu, Bo Du, and Dacheng Tao. 2022. Improving sharpness-aware minimization with fisher mask for better generalization on language models. *arXiv preprint arXiv:2210.05497* (2022).

[66] Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. 2022. Conditional prompt learning for vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16816–16825.