KeyframeQL: A Hybrid Query Interface for Spatial-temporal Events in Video Data

Jie Jeff Xu
Georgia Institute of Technology
jxu680@gatech.edu

Taehyeok Jang
Georgia Institute of Technology
tjang31@gatech.edu

Tica Lin
Dolby Laboratories
tica.lin@dolby.com

Kexin Rong
Georgia Institute of Technology
krong@gatech.edu

Abstract

Understanding complex video events requires analyzing spatio-temporal relationships across frames. Existing video database management systems (VDBMS) use either no-code interfaces (e.g., natural language, sketches) that are flexible but ambiguous or programmatic interfaces (e.g., SQL-like queries, scene graphs) that are precise but rigid. We introduce KeyframeQL, a novel query interface that combines the flexibility of no-code approaches with the precision of programmatic methods. Inspired by keyframe animation, KeyframeQL allows users to specify critical object states as keyframes while supporting rich temporal operators based on Signal Temporal Logic. Additionally, it incorporates robustness-aware predicate evaluation to handle noisy video data. We demonstrate KeyframeQL's effectiveness in traffic and sports analytics, showcasing its potential as an intermediate representation for iterative, human-in-the-loop query refinement.

ACM Reference Format:
Jie Jeff Xu, Taehyeok Jang, Tica Lin, and Kexin Rong. 2025. KeyframeQL: A Hybrid Query Interface for Spatial-temporal Events in Video Data. In . ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction and Related Work

With the rapid growth of video data across domains such as autonomous vehicles (AVs), robotics, and surveillance, there is an increasing need to efficiently extract insights from this data. Object tracking models can convert raw video into semantically rich sequences of bounding boxes and segments over time. The real value of video data, however, lies not in individual frames but in understanding the spatial-temporal relationships between them. Complex events—such as actions and interactions—can only be identified when we analyze multiple frames together to capture information like velocity, direction, and object interactions.

Video Database Management Systems (VDBMS) have explored diverse query interfaces for spatiotemporal events in videos. These approaches broadly fall into two categories, no-code interfaces and programmatic interfaces, each presenting distinct trade-offs between flexibility and precision.

No-code Interfaces (Flexible but Ambiguous) These interfaces prioritize usability by leveraging user-friendly, high-level inputs such as natural language and sketches. While they lower the barrier to entry, their reliance on high-level abstractions often sacrifices precision and hinders query debugging and fine-tuning. For example, Zelda [9] leverages vision-language models (VLMs) to match natural language queries to video clips, but its results are sensitive to input prompts–queries like "car crash" versus "car rollover" may yield different outputs. Similarly, SketchQL [11, 12] enables users to draw object trajectories using drag-and-drop motion on a canvas but struggles with ambiguity in multi-object interactions–a query like "a car stops to yield to a pedestrian crossing the road" might retrieve irrelevant trajectory pairs where cars and pedestrians move perpendicularly.

Programmatic Interfaces (Precise but Rigid) These interfaces allow users to define queries through explicit predicates or user-defined functions (UDFs) on object states and relationships. For example, MIRIS [1] offer a SQL-like interface and VQPy [13] provides an object-oriented Python programming model for defining objects and their relationships. Scene-graph-based approaches like STAR Retrieval [3] and EQUI-VOCAL [15, 16] represent queries as graphs where nodes denote objects and edges represent relations, with sequences mapping to different video frames. However, translating high-level semantic events into precise predicates is challenging due to the inherent noisiness of video data caused by camera motion or inconsistent object tracking methods, often requiring significant trial and error from users.

To address these limitations, we introduce KeyframeQL, a novel query interface that combines the flexibility of no-code approaches with the precision of programmatic approaches. The design is inspired by keyframe animation, where users specify critical states (key frames) that capture important object states, while leaving the intermediate frames unconstrained. KeyframeQL differs from prior interfaces in two key ways. First, it treats temporal constraints as a first-class citizen and supports rich temporal operators that are inspired by *Signal Temporal Logic (STL)* [7]. Second, it considers the robustness of predicate satisfaction beyond binary outcomes, which helps support approximate matches over inherently noisy video data. We envision KeyframeQL serving as an intermediate representation that helps decompose imprecise, high-level events into compositional subtasks that can be fine-tuned individually and

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

executed over video frames. This paves the way for a human-in-the-loop workflow where users can iteratively refine queries through a combination of hard constraints and soft heuristics/preferences.

In the rest of the paper, we describe the design of KeyframeQL (§ 2), share preliminary results applying KeyframeQL to traffic and sport analytics domains (§ 3), and highlight opportunities for future work (§ 4).

2 KeyframeQL Overview

KeyframeQL builds upon the scene-graph-based DSL proposed in VOCAL-UDF [16], which models videos as frame sequences represented by scene graphs of objects and their relationships. While we adopt a similar scene graph representation for individual frames, KeyframeQL enhances temporal querying through three key features: (1) inter-frame constraints for object movement and dynamic properties, (2) STL-inspired temporal operators, and (3) non-binary predicate satisfaction scoring.

2.1 Data Model and Query Language

A video consists of N frames $\{f_1, ..., f_n\}$, with clips defined as contiguous segments $\{f_i, f_{i+1}, \dots f_j\}$. Each frame contains objects (including the frame itself as a special object) and relationships that connect two or more objects. Objects have two types of properties:

- Static properties (immutable): e.g., color and type of the object
- Dynamic properties (time-varying): e.g., speed of the object

Relationships that can be either:

- Intra-frame: Connections between objects within a single frame (e.g., "pedestrian near crosswalk").
- Inter-frame: Temporal relationships that span multiple frames (e.g., tracking a car's speed changes across frames 10-20)

KeyframeQL uses a keyframe-based approach where queries are specified as a sequence of important states with constraints between them. A query consists of:

$$query = \langle \{kf_1, kf_2, ..., kf_n\}, \Phi_{inter} \rangle$$

where each keyframe kf_i is defined as:

$$kf_i = \langle \{o_1, o_2, ...\}, \Phi_{intra} \rangle$$

Specifically, keyframes $\{kf_1, kf_2, ..., kf_n\}$ represent critical states in an event sequence. Keyframes must occur in the specified order but have flexible timing – each keyframe may persist across multiple video frames, with arbitrary frames permitted between them. A keyframe contains a set of objects that must be present, as well as intra-frame constraints (Φ_{intra}) on object attributes (e.g., car.color = "red") or relationships (e.g., dist(car, pedestrian) < 5).

Inter-frame predicates (Φ_{inter}) operate across multiple frames and come in two forms:

- Object dynamics constraints track how object states change over time, such as object trajectory sketches (e.g., "car turning right") or property changes ($kf_1.o1.velocity < kf_2.o1.velocity$).
- Temporal logic operators provide control over event timing using STL-inspired constructs:
  - $always_{[t]}\phi$ requires ϕ to hold continuously for t frames.
  - $eventually_{[t]}\phi$ requires ϕ to occur within t frames (used for time gaps).
- Logical combinations ($\wedge, \vee, \neg$) enables more complex constraints. For example, $\neg(always_{[t]}kf_1)$ emualtes a *never* constraint, meaning that kf_1 should not happen within those t frames. $(kf_1 \wedge eventually_{[t]}kf_2)$ specifies that kf_2 occurs within t frames after kf_1.

2.2 Predicate Scoring and Search

Using binary True/False values for constraint evaluation could yield limited results when checking for multiple constraints, especially considering real-world video noise issues such as detection flickering, where object detections may appear and disappear across frames.

Instead, KeyframeQL uses continuous predicate scoring where each predicate returns a normalized satisfication score $p_i \in [0, 1]$, where $p_i = 1$ indicates full satisfaction, $p_i = 0$ indicates complete violation, and intermediate values indicate partial satisfaction.

The scoring system can tailor its implementation to different predicate types. For threshold-based conditions (e.g., $x > \tau$), we can use fuzzy membership functions [2] (e.g., trapezoid) to smoothly transition scores from 1 to 0 around the threshold. For semantic-based predicates (e.g., trajectory matching), KeyframeQL uses embedding space metrics such as cosine similarity between learned representations for scoring.

KeyframeQL queries are currently processed through a simple two-stage approach that first filters by hard constraints and then ranks by predicate scores - we leave more advanced query execution methods for future work. We begin by sampling video frames at regular intervals. For each sampled frame f_i, we verify whether all hard constraints in the first keyframe are fully satisfied. Successful matches trigger a temporal search, with each new keyframe being evaluated only within a valid temporal window relative to the previously matched frame. This process continues until either all keyframes are matched (yielding a valid candidate clip) or any predicate fails (discarding the candidate). For the surviving clips, the system then computes soft predicate scores that capture nuanced aspects like spatial relationships and trajectory similarities. These scores are aggregated across keyframes (e.g., averaging) to produce a final quality metric for ranking.

3 Preliminary Results

In this section, we evaluate the precision and expressiveness of KeyframeQL with spatial-temporal queries across two domains: traffic camera analysis and sports analytics. We additionally compare KeyframeQL's structured query approach against vision-language models (VLMs) with natural language interfaces.

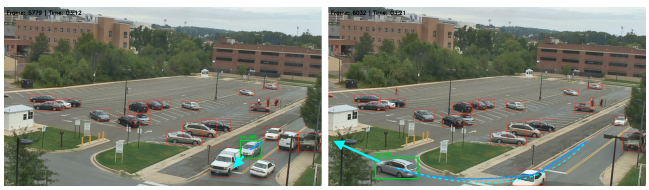


Figure 1: An example search result for query Q1.1 showing a car that is first stopped and then follows a right turn trajectory (dotted line).

3.1 Traffic Camera Analytics

Our traffic analysis utilizes two video samples from the VIRAT dataset [8]. The first captures a 3.5-minute parking lot scenario with mixed stationary and moving vehicles, while the second records four minutes of dense traffic flow at a busy urban intersection. We preprocess these videos using ByteTrack [17] for object tracking, extracting bounding box coordinates and object classifications (focusing specifically on cars). From these detections, we derive three key attributes:

- AREA (static) - the area of an object's bounding box.
- VELOCITY (dynamic) - the velocity in pixels/sec calculated by the displacement of a bounding box's centroid between consecutive frames divided by the time interval (1/FPS).
- DIR (dynamic) - the velocity unit vector.

Query Specifications. We implement two queries that capture complex vehicle behaviors:

Q1.1 (Stationary before right turn) searches for cars that are first stationary for a fixed time before taking a right turn.

Q1.2 (Convoy) searches for cars grouped tightly together, moving at the same speed and direction.

Listing 1: Q1.1: Car stopped before turing right

```
Objects:
    c1.type = 'car'

Keyframe k1: # c1 is mostly not moving
    return c1.velocity < 5

Keyframe k2: # c1 is moving
    return c1.velocity > 10

Interframe_Constraints:
    ALWAYS[k1.t, k1.t + 2.0s](k1.predicates, tol=0.5)
    ALWAYS[k2.t, k2.t + 1.0s](k2.predicates, tol=0.5)
    Trajectory(start=k1, end=k2, sketch=...) #
        Trajectory (right turn) soft constraint
```

For **Q1.1**, we construct a two-keyframe query to identify individual vehicles that remain stationary for two seconds at keyframe k_1 before initiating a turning maneuver at keyframe k_2 (Listing 1). To capture the "stopped" condition in k_1, we implement a constraint that limits the vehicle's velocity to under five pixels/second for a continuous two-second interval using the ALWAYS operator with a tolerance parameter of 0.5. This tolerance accommodates measurement noise by allowing the velocity constraint to be violated in up to 50% of frames during this period. Similarly, the "moving" condition in k_2 is specified to ALWAYS occur over a one-second duration with a 0.5 tolerance value, specified with the keyword *tol*. To capture the turning motion, we sketch a right-turn trajectory (illustrated by the dotted blue line in Figure 1) that spans between keyframes k_1 and k_2.

For **Q1.2**, we query for convoys of three vehicles traveling in formation, characterized by close proximity and directional alignment (Listing 2). The query calculates a satisfaction score for each frame based on two soft metrics: spatial compactness and directional uniformity. For both metrics, we implement the `soft_lt` fuzzy membership function that produces normalized scores between 0 and 1, with perfect scores awarded for ideal formations (e.g., small area and angular deviations under 30). The overall clip score is computed through aggregation of frame scores using the ALWAYS operator with a tolerance threshold of 0.5.

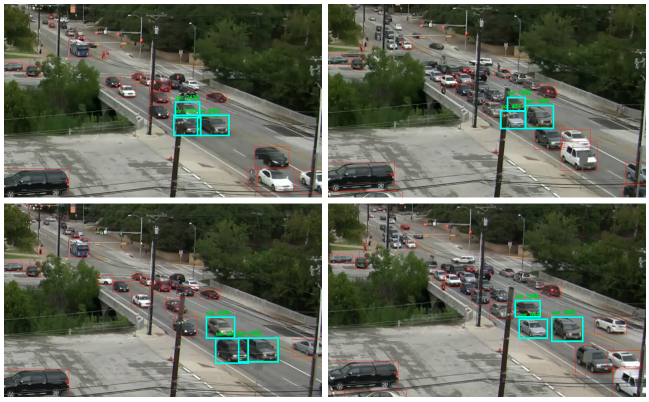


Figure 2: The top search results for example query Q1.2 featuring the best result (left column) and the 10-th best (right column).

Listing 2: Q1.2: Convoy of three cars

```
Objects:
    c1.type = 'car', c2.type = 'car', c3.type = 'car'

Keyframe k1:
    # Area constraint
    bounds_area = min_box(c1.pos, c2.pos, c3.pos).area
    expected_area = (c1.area + c2.area + c3.area)
    area_conf = soft_lt(bounds_area,
         target=expected_area)
    # Angle constraints
    avg_angle = avg(angle(c1.dir, c2.dir),
        angle(c2.dir, c3.dir), angle(c1.dir, c3.dir))
    angle_conf = soft_lt(avg_angle, target=0)
    return avg(area_conf + angle_conf)

Interframe_Constraints:
    ALWAYS[k1.t, k1.t + 2.0s](k1.predicates, tol=0.5)
```

Search Results. We present a top search result for the stop and turn query **Q1.1** in Figure 1, featuring a vehicle that remains stationary for one second before executing a right turn (depicted by the dotted light blue line). This example successfully satisfies both the hard constraints of stopping and moving, as well as the soft constraints defined by the trajectory sketch.

For the convoy query **Q1.2**, we present both the highest-ranked and tenth-ranked video clips in Figure 2. In the top-ranked clip, the vehicles maintain excellent proximity throughout the sequence, resulting in a high satisfaction score. In contrast, the tenth-ranked clip scores lower because the vehicles fail to maintain compact formation over time, demonstrating how our ranking system effectively differentiates between varying degrees of constraint satisfaction.

3.2 Sports Analytics

For sports analytics, we use the *SoccerNet-v2* action-spotting subset [4], which consists of full-length professional soccer matches (e.g., England EFL games) annotated with fine-grained action labels. We preprocess videos using object detection and tracking, team clustering, and homography-based field projection [6, 10, 14, 17] and extract the following key object properties:

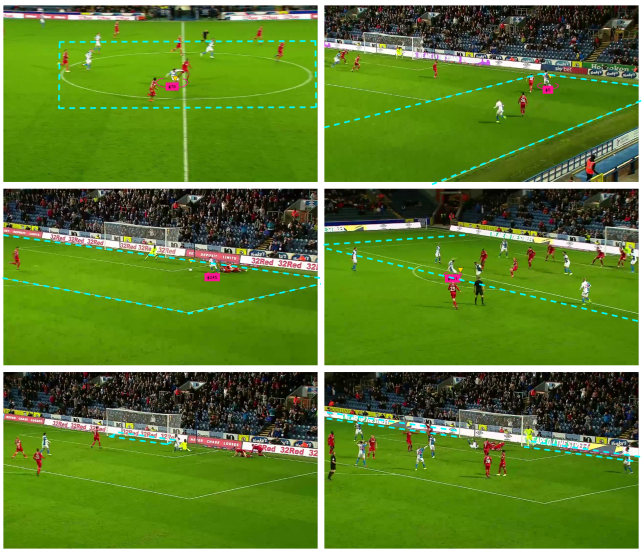


Figure 3: The search results for example queries Q2.1 (left column) and Q2.2 (right column), with each result comprising a sequence of keyframes that satisfies the specified predicates.

- TYPE (static): object class (e.g., ball, player from Team A/B).
- POS (dynamic): object's spatial coordinates projected onto the soccer field.
- ACTION (dynamic): derived from SoccerNet labels, where we merge Pass, High Pass, Cross, and Shot into a single Kicking action attributed to the player nearest to the ball. Alternatively, this attribute could also be extracted with an action classification model.

Query Specifications. We implement two goal-related queries:

Q2.1 (Long Pass to Goal) identifies sequences where a long pass from midfield quickly leads to a shot and a goal.

Q2.2 (Cross-Pass, Multiple Shots, No Goal) retrieves sequences where multiple shots occur following a cross-pass without scoring a goal.

For **Q2.1** (Listing 3), the three keyframes capture a pass from the center (k_1), a subsequent shot inside the penalty area (k_2), and the ball entering the goal (k_3). We use soft predicate scoring for goalpost proximity in k_3 to handle occasional missing ball detections. To ensure that the long pass does not get intercepted by other players, the auxiliary keyframe k_0 ensures that no other player performs a kicking action for 4 seconds after the pass starts (k_1).

For **Q2.2** (Listing 4), the keyframes describe a pass from the left or right wing (k_1), a shot near the penalty area (k_2), and subsequent shots without scoring a goal (k_3). A soft threshold is applied at k_2 to accommodate projection noise in player localization. To enforce the absence of a goal, the auxiliary keyframe k_0 ensures that no shot results in a goal within ten seconds, with tolerance for occasional false-positive ball detections.

Listing 3: Q2.1: Long Pass Leading to Goal

```
Objects:
  a1.type = 'player(A)', a2.type = 'player(A)'
  any.type = 'player', any != a1, any != a2
  ball.type = 'ball'

  center_area = bbox(x, y, w, h)
  pentaly_area_r = bbox(...)
  goal_post = bbox(...)

Keyframe k1: # a1 (center) passes to a2 (right)
  return within(a1, center_area) AND
    a1.action = 'kicking'

Keyframe k2: # a2 drives forward and shoots
  return within(a2, pentaly_area_r) AND
    a2.action = 'kicking'

Keyframe k3: # the shot leads to goal
  return soft(within(ball, goal_post))

Keyframe k0: # no interruption during pass
  return any.action = 'kicking'

Interframe_Constraints:
  ALWAYS[k1.t, k1.t + 4.0s](NOT(k0.predicates), tol=0.2)
      # no interruption (kicking by others) for 4
      seconds after k1.
  EVENTUALLY[k1.t, k1.t + 10.0s](k3.predicates)
  EVENTUALLY[k2.t, k2.t + 1.5s](k3.predicates)
```

Listing 4: Q2.2: Cross-Pass Leading to Multiple Shots (No Goal)

```
Objects:
  a1.type = 'player(A)'
  a2.type = 'player(A)'
  any.type = 'player', any != a1, any != a2
  ball.type = 'ball'

  left_wing = bbox(x1, y1, w, h)
  right_wing = bbox(...)
  pentaly_area_r = bbox(...)
  goal_post = bbox(...)

Keyframe k1: # a1 (wing) passes to a2 (penalty area)
  return within(a1, left_wing OR right_wing) AND
     a1.action = 'kicking'

Keyframe k2: # a2 drives into the penalty area and shoots
  return soft(within(a2, (pentaly_area_r + 50))) AND
    a2.action = 'kicking'

Keyframe k3: # a3 subsequently shoots
  return a3.action = 'kicking'

Keyframe k0: # no shot leads to goal
  return within(ball, goal_post)

Interframe_Constraints:
  EVENTUALLY[k1.t, k1.t + 4.0s](k2.predicates)
  EVENTUALLY[k2.t, k2.t + 1.5s](k3.predicates)
  ALWAYS[k2.t, k2.t + 10.0s](NOT(k0.predicates), tol=0.1)
```

Search Results. The search results are visualized in Figure 3. KEYFRAMEQL successfully located both ground truth clips ([03:53–04:01] for Q2.1 and [03:19–03:25] for Q2.2) that were manually annotated by the authors. The retrieved clips demonstrate robustness to noise. Specifically, the predicate scoring successfully handled a missing ball detection at keyframe k_3 in Q2.1 and a wrong projection along the y-axis at keyframe k_2 in Q2.2. We also observed some near-duplicate results in the retrieved clips, which occurred due to tracking inconsistencies where different IDs were assigned to the same physical objects. These duplicates were subsequently removed using simple post-processing steps.

3.3 Comparison with Vision–Language Model

We compare KeyframeQL against a strong "no-code interface" baseline—Google Gemini 2.0-Flash [5]—on the same video segments and query intents introduced in this section. As a state-of-the-art multimodal model, Gemini represents the closest practical alternative for users who prefer natural language queries over structured compositional queries. Table 1 provides a qualitative comparison.

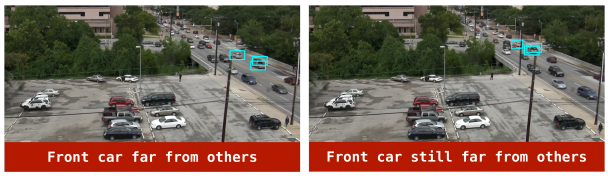
Overall, across all prompted queries in both domains, KeyframeQL demonstrates higher retrieval precision compared to Gemini. Specifically, KeyframeQL consistently retrieves clips that satisfy the spatial-temporal predicates, whereas Gemini often returns clips that are broadly relevant to keywords (e.g., shot, goal) but may hallucinate unrelated or only partially matching events.

In handling spatio-temporal constraints, KeyframeQL successfully retrieves clips that satisfy explicit spatial and temporal relationships between keyframes as seen in Figure 3. In contrast, Gemini struggles to accurately capture spatial constraints and fine-grained temporal sequences, even when such conditions are explicitly described in the prompt. For instance, in the soccer query (Q2.2), Gemini fails to retrieve clips that satisfy nuanced spatial constraints such as a pass from the right wing, or fine-grained temporal sequences such as multiple shots occurring over time.

Moreover, KeyframeQL uses continuous predicate scoring to assess the degree of satisfaction for each spatial, semantic, and temporal constraint, enabling a more nuanced evaluation of matched clips. In contrast, Gemini lacks fine-grained predicate scoring and often returns results that violate temporal constraints without providing rationale or confidence signals. This difference is particularly evident in the convoy query for the traffic camera domain (Q1.2), where KeyframeQL effectively prioritizes tighter vehicle groups by scoring spatial compactness, as demonstrated in Figure 2. In contrast, Gemini fails to identify truly compact convoys or rank them appropriately according to compactness criteria.

Consistent with observations from prior works, we also notice that Gemini performance is sensitive to input prompts–slightly altered queries (e.g., *center* vs. *right-wing*, or *goal* vs. *no goal*) can give drastically different results.

Q1.2: Convoy of three cars



Q2.2: Cross→Multiple shots (No Goal)

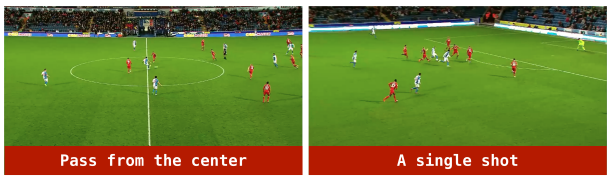


Table 1: Visualized query results retrieved by Google Gemini 2.0-Flash, used as a baseline no-code interface.

4 Challenges and Future Work

While KeyframeQL shows promising initial results in retrieving complex spatial-temporal events, several key challenges remain to improve its usability and efficiency. We highlight promising research directions to address these gaps:

Human-in-the-Loop Query Refinement. Our preliminary results suggest that iterative refinement is essential—many retrieved clips may only partially satisfy the specified predicates, making user feedback crucial for refining retrieved results. We believe that compared to unstructured natural language prompts, KeyframeQL facilitates more efficient iterations and refinement of queries through its combination of hard and soft constraints. An important research question is how to best help users refine query specifications based on retrieved results. For example, the system could request refinement of too loose or strict constraints, provide visual feedback on predicate satisfaction levels, or use active learning techniques to prioritize user feedback on ambiguous query intent.

Intuitive Query Specification. We recognize that query specification with KeyframeQL is still time-consuming, as users need to manually define a threshold in predicates as well as implementations of UDFs. We envision multiple approaches to reduce this barrier to entry. One potential direction leverage users' familiarity with video editing software to develop a visual query interface. Similar to SketchQL, users could create objects, edit their properties, and specify trajectories by dragging and dropping elements on a canvas. The interface would allow inserting keyframes along a timeline and specifying both intra-frame and inter-frame constraints by selecting one or more keyframes. Another promising direction is to explore using Large Language Models (LLMs) to automatically synthesize UDF implementations based on natural language descriptions, such as generating appropriate fuzzy membership functions for specific constraints. The system could also provide smart defaults, such as suggesting temporal tolerance thresholds based on the noise characteristics of the input videos.

Query Execution Efficiency. Our prototype currently preprocesses entire videos to extract relevant object attributes, which can be inefficient since most video content may not be relevant to a particular query. Future work can explore on-the-fly and sampling-based preprocessing to reduce computational costs. In addition, our search algorithm iterates over all combinations of objects to check predicate satisfaction, which can be challenging in domains like soccer analytics with many objects of interest. While our example queries mitigate this by grouping objects (e.g., Team A vs. Team B) and focusing on ball-possessing players, broader use cases would benefit from a more general optimization framework. This could incorporate techniques such as predicate reordering to evaluate cheaper constraints first, spatial indexing to quickly eliminate irrelevant object combinations, and pruning strategies based on temporal or spatial coherence. Furthermore, implementing caching mechanisms can further enhance performance as users iteratively refine their predicate filters. By storing intermediate results and predicate satisfaction scores, KeyframeQL can avoid redundant computation when users adjust their queries.

References

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

[2] Guanrong Chen and Trung Tat Pham. 2005. *Introduction to fuzzy systems*. CRC Press.

[3] 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.

[4] Adrien Deliège, Anthony Cioppa, Silvio Giancola, Meisam J. Seikavandi, Jacob V. Dueholm, Kamal Nasrollahi, Bernard Ghanem, Thomas B. Moeslund, and Marc Van Droogenbroeck. 2021. SoccerNet-v2 : A Dataset and Benchmarks for Holistic Understanding of Broadcast Soccer Videos. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

[5] Google. 2024. The Gemini era: Bringing the power of Gemini 1.5 to more products and developers. https://blog.google/technology/google-deepmind/google-gemini-ai-update-december-2024 Accessed: 2025-04-13.

[6] Abiodun M Ikotun, Absalom E Ezugwu, Laith Abualigah, Belal Abuhaija, and Jia Heming. 2023. K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. *Information Sciences* 622 (2023), 178–210.

[7] Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*. Springer, 152–166.

[8] Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, Jake K 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.

[9] Francisco Romero, Caleb Winston, Johann Hauswald, Matei Zaharia, and Christos Kozyrakis. 2023. Zelda: Video analytics using vision-language models. *arXiv preprint arXiv:2305.03785* (2023).

[10] Ultralytics. 2021. YOLOv5: A state-of-the-art real-time object detection system. https://docs.ultralytics.com. Accessed: Marched 2025.

[11] Renzhi Wu, Pramod Chunduri, Ali Payani, Xu Chu, Joy Arulraj, and Kexin Rong. 2024. SketchQL: Video Moment Querying with a Visual Query Interface. *Proc. ACM Manag. Data* 2, 4, Article 204 (Sept. 2024), 27 pages. doi:10.1145/3677140

[12] Renzhi Wu, Pramod Chunduri, Dristi J Shah, Ashmitha Julius Aravind, Ali Payani, Xu Chu, Joy Arulraj, and Kexin Rong. 2024. SketchQL Demonstration: Zero-Shot Video Moment Querying with Sketches. *Proc. VLDB Endow.* 17, 12 (Aug. 2024), 4429–4432. doi:10.14778/3685800.3685892

[13] Shan Yu, Zhenting Zhu, Yu Chen, Hanchen Xu, Pengzhan Zhao, Yang Wang, Arthi Padmanabhan, Hugo Latapie, and Harry Xu. 2024. VQPy: An Object-Oriented Approach to Modern Video Analytics. *Proceedings of Machine Learning and Systems* 6 (2024), 279–295.

[14] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. 2023. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF international conference on computer vision*. 11975–11986.

[15] Enhao Zhang, Maureen Daum, Dong He, Brandon Haynes, Ranjay Krishna, and Magdalena Balazinska. 2023. Equi-vocal: Synthesizing queries for compositional video events from limited user interactions. *Proceedings of the VLDB Endowment* 16, 11 (2023), 2714–2727.

[16] Enhao Zhang, Nicole Sullivan, Brandon Haynes, Ranjay Krishna, and Magdalena Balazinska. 2024. Self-Enhancing Video Data Management System for Compositional Events with Large Language Models [Technical Report]. *arXiv preprint arXiv:2408.02243* (2024).

[17] 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 *European conference on computer vision*. Springer, 1–21.