MIRIS: Fast Object Track Queries in Video

Favyen Bastani, Songtao He, Arjun Balasingam
Karthik Gopalakrishnan, Mohammad Alizadeh, Hari Balakrishnan
Michael Cafarella, Tim Kraska, Sam Madden
Massachusetts Institute of Technology
{favyen,songtao,arjunvb,karthikg,alizadeh,hari,michjc,kraska,madden}@csail.mit.edu

ABSTRACT

Video databases that enable queries with object-track predicates are useful in many applications. Such queries include selecting objects that move from one region of the camera frame to another (e.g., finding cars that turn right through a junction) and selecting objects with certain speeds (e.g., finding animals that stop to drink water from a lake). Processing such predicates efficiently is challenging because they involve the movement of an object over several video frames. We propose a novel query-driven tracking approach that integrates query processing with object tracking to efficiently process object track queries and address the computational complexity of object detection methods. By processing video at low framerates when possible, but increasing the framerate when needed to ensure high-accuracy on a query, our approach substantially speeds up query execution. We have implemented query-driven tracking in MIRIS, a video query processor, and compare MIRIS against four baselines on a diverse dataset consisting of five sources of video and nine distinct queries. We find that, at the same accuracy, MIRIS accelerates video query processing by 9x on average over the IOU tracker, an overlap-based tracking-by-detection method used in existing video database systems.

ACM Reference Format:
Favyen Bastani, Songtao He, Arjun Balasingam, Karthik Gopalakrishnan, Mohammad Alizadeh, Hari Balakrishnan, and Michael Cafarella, Tim Kraska, Sam Madden. 2020. MIRIS: Fast Object Track Queries in Video. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3318464.3389692

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3389692

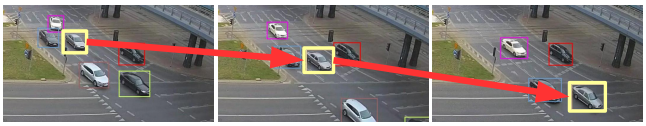

Figure 1: An example track over three video frames.

1 INTRODUCTION

Automated analysis of video data has become crucial to an ever-expanding range of applications, from traffic planning [20] to autonomous vehicle development [8]. A common class of video analysis involves queries over *object tracks*, which are sequences of object detections corresponding to the same object instance (Figure 1). These queries select instances of a particular object category (e.g., cars, traffic signs, animals) through predicates on their trajectories over the segments of video in which they appear. As an example, an autonomous vehicle engineer debugging anomalous driving behavior under particular conditions may wish to query previously collected video data to select segments where those conditions appeared. Such a query likely applies a predicate over object tracks; e.g., selecting video where pedestrians walk in front of the car involves a predicate on pedestrian tracks that pass from one side of the camera frame to the other.

In general, object track queries may apply complex predicates, including joins that involve predicates over multiple temporally overlapping tracks. For example, we may express the task of identifying situations where a car passes a cyclist at high speed and close proximity with the query:

```
SELECT car, cyclist FROM (PROCESS inputVideo
      PRODUCE car, cyclist USING objDetector)
WHERE Speed(car) > 30 km/h
AND Angle(car, cyclist) < 10 deg
AND MinDistance(car, cyclist) < 1 m
```

This query first applies object detection and object tracking to derive tracks of cars and cyclists in the video dataset. It includes three user-defined function (UDF) predicates to verify that the car is moving at high speed, the car and cyclist

are traveling in the same direction, and that they pass within one meter of each other.

Video analytics databases such as DeepLens [14] and Rekall [9] execute these queries by applying object detection and tracking over the entire video, and then selecting the tracks that satisfy the query predicates. However, executing the deep neural networks used in object detection is highly compute-intensive. On the $10,000 NVIDIA Tesla V100 GPU, the YOLOv3 object detector [18] can process 960×540 video frames at 30 frames per second (fps). Thus, applying object detection on every frame in large video datasets is expensive—processing one month of video captured at 10 fps from 100 cameras (72K hours of video) would cost over $70,000 on Amazon Web Services (AWS) today (using a p3.2xlarge AWS instance type at $3.06/hr for 24,000 instance-hours).

Recently, several video query processing engines have been proposed to accelerate the execution of video analytics queries by addressing the GPU-intensiveness of object detection. Broadly, these engines, which include NoScope [13], probabilistic predicates [16], BlazeIt [12], and SVQ [21], train lightweight, specialized machine learning models to approximate the result of a predicate over individual video frames or sequences of frames. If the lightweight model is confident that a segment of video does not contain any object instances that satisfy the predicate, the query processor skips expensive object detector execution over the video segment.

These optimizations are effective for processing queries with only per-frame predicates, such as identifying video frames containing at least one bus and one car. However, two issues limit their applicability to object track queries. First, oftentimes, object instances that satisfy the query predicate appear in almost every frame, e.g. when finding cars that turn through a junction. Since expensive object detectors must be executed for each positive result from the lightweight model to confirm the predicate output, the lightweight model does not offer a substantial speedup. Second, object track queries inherently involve sequences of video, since predicates operate on the trajectories of objects through the camera frame. Although we can train a lightweight model to input entire segments of video, we will show that in practice this approach yields models with low accuracy (and thus low speedups). Thus, even after applying these optimizations, existing systems must still apply expensive object detectors on almost every video frame to process object track queries.

Rather than accelerate query execution through lightweight approximations of query predicates, we propose optimizing execution speed on a different dimension: the framerate at which we sample video during query execution. If we can accurately track objects and evaluate query predicates while sampling video at a framerate lower than the original capture rate (e.g., sampling at 1 fps instead of 25 fps), then we can substantially reduce the overall cost of executing object track queries. In practice, though, determining the minimum sampling framerate at which we can still compute accurate query outputs is challenging, as it may vary significantly between different video segments. For the car-passing-cyclist query above, a low sampling framerate may be suitable during periods of light traffic, when robustly tracking a small number of cars and cyclists is straightforward. In medium traffic, though, high object densities result in uncertainty in computed object tracks, and we may need to increase the sampling framerate to reduce uncertainty. On the other hand, in heavy traffic, despite the same issue of uncertainty, because traffic speeds are low, we may be confident that no cars travel at speeds high enough to satisfy the first query predicate, and thus return to a low sampling framerate.

To address this challenge, we propose a novel optimization, *query-driven tracking*, that integrates query processing and object tracking to select a variable video sampling framerate that minimizes object detector workload while maintaining accurate query outputs. We begin query execution by applying object detection and object tracking at a reduced framerate over the video to obtain object tracks. Due to the low sampling framerate, some tracks contain uncertainty where the tracking algorithm is not confident that it has correctly associated a sequence of detections. If we were to perform object tracking agnostic of query processing, we would immediately address this uncertainty by sampling additional video frames at a higher framerate. In our integrated execution approach, we minimize the additional sampling required by ignoring uncertainty when we are confident that the affected tracks do not satisfy the query predicate.

However, even after resolving uncertainty in the computed tracks, because low-framerate tracking produces *coarse-grained tracks* where consecutive detections along the track may be seconds apart, the query predicate may not evaluate correctly over the tracks. If, for example, a query seeks instances of hard braking before a traffic light, we may miss a hard braking event when sampling video at 0.25 fps (4 seconds/frame) if a car stops within 2 seconds. As with uncertainty in object tracking, we must process additional frames until we are confident that tracks have sufficiently fine-grained detections for accurate predicate evaluation. We develop a filtering-refinement approach that first prunes tracks that we are confident do not satisfy the predicate, and then refines the remaining tracks by collecting additional object detections along the tracks in a targeted manner.

Our approach exposes several parameters, including the minimum sampling framerate and selections of filtering and refinement methods, to generalize to a wide range of object track queries. We develop a novel query planner that selects these parameters for a specific query with the objective of maximizing query execution speed while satisfying a user-specified accuracy bound. Our query planner estimates

accuracy by evaluating an execution plan over pre-processed segments of video where we have already applied object detection and object tracking at the full video framerate.

In summary, our contributions are:

- We develop a novel query-driven tracking optimization that integrates query processing into the object tracker to minimize the number of video frames processed when executing object track queries. Our approach exposes several parameters, such as filtering and refinement methods, that can be chosen to optimize execution for a wide range of queries.
- We propose a query planning algorithm that automatically selects these parameters for each query by estimating cost over pre-processed segments of video.
- We implement our approach in the MIRIS video query processor, and evaluate it on 9 distinct object track queries over 5 diverse datasets. We find that, at the same accuracy, MIRIS accelerates query execution by 9x on average over the overlap tracking-by-detection method [4] used in existing video database systems such as BlazeIt [12] and Rekall [9]. The MIRIS source code is available at https://github.com/favyen/miris.

2 RELATED WORK

Several systems have recently been proposed for processing queries over large volumes of video data. NoScope [13] applies a cascaded detection approach to quickly identify video frames containing instances of a specific object type, e.g., frames containing a car. It first applies weak classifiers including image differencing and shallow convolutional neural networks (CNNs) on each frame, and only applies expensive but high-accuracy deep CNNs on frames where the weak classifiers have low confidence. BlazeIt [12] and SVQ [21] extend the use of shallow CNNs for a variety of other per-frame predicates by combining specialized CNN training with approximate query processing techniques. For instance, these systems may process a query seeking the average number of cars that appear in video frames by training a lightweight CNN to output a car count on each frame.

However, while existing work on video query optimization support queries involving aggregation over entire videos, they focus on predicates that can be evaluated over individual frames. Although probabilistic predicates [16] extends specialized classifiers for predicates over sequences of frames, we will show in our evaluation that this approach is not effective for most object track queries. Additionally, prior optimization approaches do not accelerate queries where instances of the object type of interest appear in every frame.

Indeed, this gap has prompted the development of new video query processors such as Rekall [9] that simply assume that object detections are available in every video frame in order to provide query languages that are sufficiently expressive for practical applications. Although these systems enable a substantially wider range of applications, they require executing object detection models over all video data being queried, which is costly for large video datasets.

Approaches for resource-efficient object detection and tracking have also been studied outside of the video query processing context. Several approaches apply correlation filters to efficiently track individual objects over video given the object's position in an initial frame [11, 17]. Deep Feature Flow [23] proposes speeding up object detection in video by applying a deep CNN on key frames (e.g., one in ten frames), and using a lightweight FlowNet [5] CNN to propogate detection outputs across intermediate frames. As we will show in our evaluation, because these methods are query-agnostic, they offer a poor tradeoff between speed and accuracy for processing object track queries.

3 OVERVIEW

In this section, we provide an overview of MIRIS and its query execution process.

3.1 Queries

MIRIS provides a declarative interface for querying video data to select object tracks of a certain category (e.g., car, pedestrian, animal) that satisfy user-defined predicates. An *object track* $A = \langle d_{i_1}^{k_1}, \ldots, d_{i_n}^{k_n} \rangle$ is a sequence of temporally-ordered *object detections* that correspond to the same object over the segment of video in which the object is visible in the camera frame. Each object detection $d_{i_j}^{k_j}$ in the track specifies a bounding box in the (k_j)-th video frame, $d_{i_j}^{k_j} = (x_{i_j}^{k_j}, y_{i_j}^{k_j}, w_{i_j}^{k_j}, h_{i_j}^{k_j})$, where $(x_{i_j}^{k_j}, y_{i_j}^{k_j})$ is the center of the bounding box and $(w_{i_j}^{k_j}, h_{i_j}^{k_j})$ is its width and height.

A predicate P produces a boolean output given one or more tracks $(A_1, \ldots, A_m)$. Queries that select individual tracks consist of boolean combinations of geometric predicates over the position and speed of a track. For example, a query that selects car tracks that rapidly decelerate before a traffic junction would include a predicate that computes the acceleration over a track and thresholds the maximum deceleration.

Queries may also select tuples of object tracks satisfying predicates over multiple temporally overlapping tracks (joins). The car-passing-cyclist example in the introduction applies two predicates over both the car and cyclist tracks requiring that the two tracks be traveling in the same direction and pass within 1 m of each other.

In general, queries in MIRIS may employ arbitrary UDF predicates over one or more tracks. To ensure applicability to a wide range of object track queries, our query-driven tracking algorithm does not assume knowledge of the predicate

formulation — instead, we train a machine learning model to predict whether a given coarse track satisfies a predicate P based on its boolean outputs over tracks in a pre-processed segment of video, and use this and other models to make decisions in our optimizations during query execution.

3.2 Pre-processing

When MIRIS ingests new video, it first pre-processes the video to collect known, accurate object tracks that our query planner will use as training and validation data to select optimal query-driven tracking parameters. During pre-processing, we first randomly sample N video segments (each segment containing multiple frames) of duration T uniformly over the video dataset. In each sampled segment, we execute an object detection model (YOLOv3 [18]) and object tracking algorithm (the IOU algorithm [4]) at the full framerate to compute object tracks that the system assumes are correct. We split the sampled video segments so that half are used as training data and half as validation data. We denote the set of tracks in the training segment as S_{train}, and the set in the validation segment as S_{val}.

Larger N and T entails pre-processing more video, but yield more accurate query planning decisions, which in turn speeds up query execution by enabling the planner to apply more aggressive optimizations. In practice, we find that different choices of N, T lead to similar tradeoffs across different queries and video data; e.g., $N = 24$, $T = 5$min is effective for the queries we consider in Section 7. Nevertheless, if planning produces an execution plan with a cost estimate that is slower than the user desires, MIRIS can return to the pre-processing phase and sample additional video segments.

MIRIS targets queries over large datasets with thousands of hours of video, so pre-processing time is negligible. Additionally, MIRIS only pre-processes each video dataset once.

When executing queries involving rare events that cannot be expressed as a conjunction of multiple more common predicates, there may be zero or only a few instances of tracks satisfying the query predicate in the pre-processed segment of video. This limits the optimizations that the planner can apply. For these queries, we begin query execution at a slower unoptimized speed, and the user re-executes the planner after sufficient video has been processed.

3.3 Query Processing

When a user runs a query, MIRIS first plans query execution by selecting parameters, including the minimum sampling framerate at which we will execute object tracking and combinations of filtering and refinement methods. The objective during planning is to minimize expected query execution time under a user-specified accuracy bound (e.g, 99%). We detail planning in Section 5. Once planning completes, we execute the selected query plan over the entire video dataset.

We detail query execution in the next section.

4 QUERY-DRIVEN TRACKING

MIRIS integrates query processing and object tracking to substantially boost query execution speed by varying the sampling framerate over the video dataset. In segments of video where we can accurately track objects and evaluate the query predicate while processing video at a reduced framerate, we do so. In other video segments, though, we may process additional video frames at a higher framerate to resolve uncertainty in object tracking decisions or to ensure that predicates can be accurately evaluated over the coarse tracks produced by low-framerate object tracking.

Our query-driven tracking approach operates in four stages: tracking, filtering, uncertainty resolution, and refinement. We summarize these stages in Figure 2.

Tracking. Initially, we track objects at a minimum sampling framerate selected by the query planner. We develop a graph neural network (GNN) tracking model that inputs visual and spatial features describing object detections across two consecutive frames, and outputs the probability that each pair of detections are the same object. When the GNN has low confidence in a matching decision, and cannot decide between multiple potential matchings of detections, it produces a group C of candidate *nondeterministic tracks* corresponding to all of the matchings between detections in consecutive frames that the GNN believes are plausible. These are represented as orange edges in Figure 2. Only a subset of the nondeterministic tracks are correct. High-confidence tracks that do not involve any uncertain matching decisions form their own groups C (e.g., C_2 in the figure).

The remaining stages process tracks group-by-group.

Filtering prunes candidate tracks in a group C that we are confident will not satisfy the query predicate P. It outputs a set $filter(C)$ containing only tracks in C that may satisfy P. If $filter(C)$ is empty, we can immediately stop processing C and return to tracking, since we are confident that none of the tracks in C will appear in the query outputs.

Uncertainty resolution addresses the remaining nondeterministic tracks in $filter(C)$. It resolves uncertain matching decisions by recursively processing additional video frames around each such decision to determine which of the nondeterministic matchings are correct. This procedure yields a subset $resolve(C) \subset filter(C)$ of deterministic tracks consisting of each track in $filter(C)$ that we find corresponds to correct matching decisions.

Refinement. Finally, we must evaluate whether each track in $resolve(C)$ satisfies P. However, since detections in

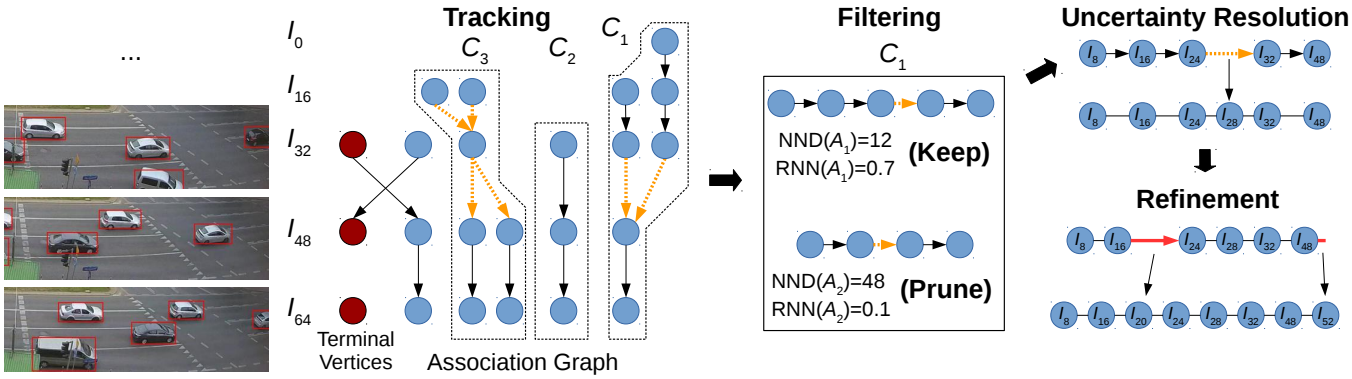

Figure 2: Our four-stage query-driven tracking approach. Orange dashed edges result from uncertain matching decisions, and yield multiple nondeterministic tracks, which are resolved during uncertainty resolution. Refinement identifies segments of tracks (red edges) that require additional detections for accurate predicate evaluation, and processes intermediate video frames in those segments.

Table 1: Summary of the notation in our approach.

f	A video sampling frequency
I_k	The k-th video frame
d_i^k	An object detection in frame I_k
$p_{i,j}^k$	Likelihood that d_i^k and d_j^{k+f} are the same object
S_{train}, S_{val}	Tracks computed in the pre-processed segments
$P(S)$	Subset of tracks in S that satisfy a predicate P
C	A group of nondeterministic tracks
$filter(C)$	Tracks in C retained through filtering
$resolve(C)$	Subset of deterministic tracks in $filter(C)$
Q_f	Uncertainty threshold at sampling frequency f

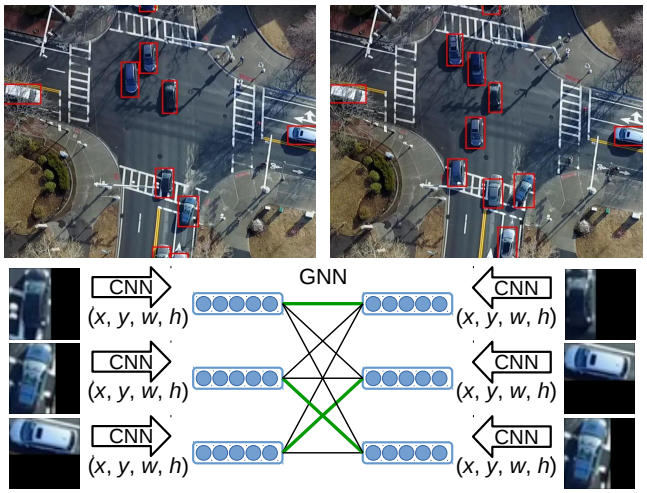

Figure 3: We match detections between two consecutive video frames sampled at low framerate by solving a bipartite matching problem between detections across the frames through the CNN and GNN model.

these tracks may be seconds apart, P may not evaluate correctly over the coarse tracks. Thus, before evaluating P, we refine the tracks by processing additional video frames along the track that we determine are needed to evaluate the predicate accurately, and extending candidate tracks in $resolve(C)$ with detections computed in these additional frames. After refinement, we evaluate P over the refined tracks and output each track where P is true.

We summarize the notation that we use in this section in Table 1. Below, we first detail the tracking, filtering, uncertainty resolution, and refinement stages for object track queries that select individual tracks satisfying a predicate P. In these stages, we present several alternative methods for filtering and refining tracks; methods that work best for a particular query are automatically chosen by the optimizer, described in Section 5.

4.1 Tracking

Accurate object tracking depends on robustly matching object detections that pertain to the same object instance between consecutive video frames. This is generally formulated as a bipartite matching problem, with object detections from the first frame on one side of the bipartite graph, and detections from the second frame on the other side. Suppose that we have applied an object detector on two video frames I_k and I_{k+f}, where f is a maximum video sampling frequency selected during planning, e.g. if $f = 2$ then we skip every other frame and thereby halve the original video capture framerate. Then, we obtain two sets of detections,

$D^k = \{d_1^k, \ldots, d_n^k\}$ and $D^{k+f} = \{D_1^{k+f}, \ldots, d_m^{k+f}\}$. Each detection is a bounding box $d_i^k = (x_i^k, y_i^k, w_i^k, h_i^k)$ in frame I_k.

We define a directed bipartite graph $G = (V, E)$ where each vertex on the left side corresponds to a detection $d_i^k \in D^k$, and each vertex on the right side corresponds to a detection $d_i^{k+f} \in D^{k+f}$. We assume that any two detections across the two frames may correspond to the same object instance; thus, the bipartite graph is densely connected. Then, the matching problem is to select the set of edges $E^* \subset E$ that connect detections of the same object instance between I^k and I^{k+f}. Note that it is possible that some detections in either frame may not match with any detection in the other frame, since objects may leave or enter the camera's field of view. Thus, we include *terminal vertices* d_0^k and d_0^{k+f} on either side of the bipartite graph. A detection d_i^k that left the camera frame should be matched to d_0^{k+f}, and a detection d_j^{k+f} that entered the camera frame should be matched to d_0^k.

Unsupervised tracking-by-detection methods such as IOU [4] and SORT [3] select edges in G based on bounding box overlap — intuitively, pairs of detections (d_i^k, d_j^{k+f}) with high overlap more likely correspond to the same object than pairs with little or no overlap. However, these methods fail when tracking at reduced framerates (high f), since detections of the same object seconds apart may not overlap at all.

4.1.1 Basic GNN Model. Thus, we instead solve the matching problem by applying a machine learning model trained to associate detections of the same object. Specifically, we use a hybrid model consisting of a convolutional neural network (CNN) and a graph neural network (GNN) [6]. We show the model architecture in Figure 3. For each object detection d_i^k, the CNN inputs the region of I_k corresponding to the detection bounding box, and computes a vector of visual features f_i^k. The GNN operates on G, and at each vertex d_i^k, inputs both the visual features f_i^k and a 4D spatial feature vector $(x_i^k, y_i^k, w_i^k, h_i^k)$. The GNN reasons about both visual and spatial features through a series of graph convolutional layers, and outputs a probability $p_{i,j}^k$ that each edge (d_i^k, d_j^{k+f}) in the graph is a correct match. The outputs include the probability $p_{i,0}^k$ that d_i^k left the camera frame, and the probability $p_{0,j}^k$ that d_j^{k+f} is a new object track. We train the GNN over pre-processed segments of video where we have computed object tracks accurately by examining every video frame; the GNN model is trained just once per object category.

4.1.2 Integrating object tracking and query processing. Were we to apply this model at the full video framerate ($f = 1$), we would compute object tracks from the edgewise probabilities through the Hungarian assignment method, which, given a cost at each edge $(1 - p_{i,j}^k)$, computes a bipartite matching that minimizes total cost. (The terminal vertices d_0^k and d_0^{k+f} must be specially handled to compute accurate assignments.) However, when applying the model at reduced framerates, this method yields occasional tracking errors where detections of distinct objects are incorrectly matched because the model has low confidence over several plausible matchings.

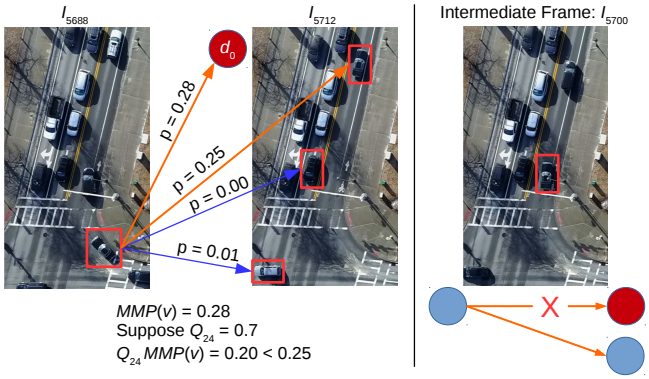

Figure 4: When matching the car on the left with vertices in I_{5712}, the GNN assigns high probabilities to two edges: one indicates that the car exited the camera frame, and one connects to the correct car. We resolve the uncertainty by examining the intermediate frame.

Instead, when the output probabilities indicate that the model has low confidence, we can examine an intermediate video frame $I_{k+f/2}$ halfway between I_k and I_{k+f} to resolve the uncertain matching decisions — the model should have higher confidence when matching video frames at higher framerates, and thereby yield greater accuracy. We define the maximum matching probability $MMP(v)$ of a vertex v in the bipartite graph G as the maximum probability over edges incident to v; for example, if $v = d_i^k$, then $MMP(d_i^k) = \max_j p_{i,j}^k$. Given a framerate-specific probability threshold $0 < Q_f \leq 1$, we define the *matching output graph* between I_k and I_{k+f} as a subgraph $\hat{G} \subset G$ containing all of the vertices in G, but only including an edge (d_i^k, d_j^{k+f}) if $p_{i,j}^k > Q_f MMP(d_i^k)$. Thus, we include in $\hat{G}$ not only the edges with highest probabilities computed through the GNN, but also any edge with an output probability that is within a factor Q_f to these highest probabilities. We say that an edge (u, v) in the matching output graph $\hat{G}$ is *nondeterministic* if u has multiple outgoing edges or v has multiple incoming edges. These cases indicate that there is at least one other detection that matches to u or v with probability close to the highest confidence matching; thus, nondeterministic edges correspond to uncertain matching decisions, where multiple matchings are plausible.

We show an example uncertain matching decision and corresponding GNN outputs in Figure 4. The GNN initially

assigns high probabilities to two edges, but the uncertainty is resolved after processing an intermediate video frame.

To keep track of candidate tracks and uncertain matchings, we build an *association graph* H by concatenating matching output graphs across a series of frames $I_0, I_f, I_{2f}, \ldots$. Each vertex in H is a detection d_i^k or a terminal vertex d_0^k, and an edge connects (d_i^k, d_j^{k+f}) if the edge appears in the matching output graph for (I_k, I_{k+f}). We show an example association graph in Figure 2. When we process additional video frames during object tracking, H grows to the bottom.

Candidate tracks are represented as paths in H that begin and end at terminal vertices. If a candidate includes any nondeterministic edge, we say that the candidate is a nondeterministic track, since we can only know whether the track is correct by processing intermediate video frames. Similarly, a deterministic track is a path that contains no nondeterministic edges.

The filtering, uncertainty resolution, and refinement stages process groups of candidate tracks that share uncertain matching decisions. These groups are connected components in H. After applying object tracking through a video frame I_k, we identify each connected component C in H that contains no vertex corresponding with I_k, i.e., C contains only candidate tracks that terminate on or before I_{k-f}. Thus, tracking over further video frames will not modify C. We remove C from H and pass it to the next three stages for further processing.

4.2 Filtering

Filtering operates on a connected component C of the association graph H produced by tracking. Pruning candidate tracks in C that we are confident will not satisfy P boosts query execution speed by reducing the number of additional video frames that must be processed during uncertainty resolution and refinement.

In our approach, we assume that the predicate formulation is unknown; however, filtering is challenging even when the formulation of P is known. Consider the example of filtering the red track in Figure 5 for a query that selects tracks that start in the left, blue region and end in the right, orange region. Given only the regions, we cannot determine with confidence whether the red track satisfies P. However, when we consider tracks from the pre-processed segment that we know satisfy P, it appears likely that the red track does not satisfy P, since it is dissimilar from the known tracks.

Thus, we develop two methods, a Nearest Neighbor Distance method and a Recurrent Neural Network (RNN), that adapt existing techniques for our filtering task. Both methods use tracks from pre-processing known to satisfy P to predict whether a candidate track during query execution satisfies P. We denote the pre-processed tracks as S_{train}, and the subset

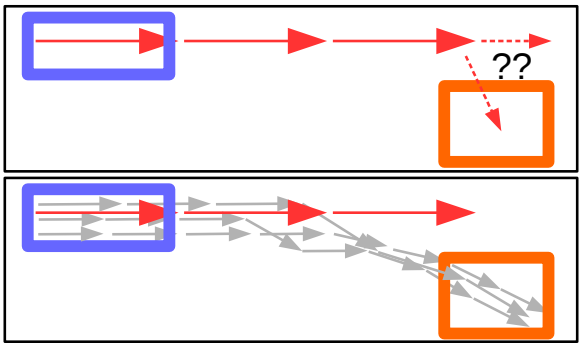

Figure 5: Illustration of the difficulty of filtering. A query selects tracks that start in the left, blue region and end in the right, orange region. Top: we cannot determine whether to prune the red track. Bottom: by comparing it to tracks seen during pre-processing (grey), we can prune with greater confidence.

that satisfy P as $P(S_{\text{train}}) \subset S_{\text{train}}$. Nearest Neighbor Distance is an unsupervised method that leverages a distance function between tracks and is effective when there are few tracks in $P(S_{\text{train}})$. The RNN filter applies a recurrent neural network for the prediction task, and provides improved prediction accuracy over a wider range of predicates when many tracks are available for training. We use a lightweight RNN model that inputs a small set of spatial features, and so we find that it is effective when trained on as few as 40 positive example tracks and converges within three minutes of training.

Both filters output a real number prediction indicating the filter's confidence that a track satisfies P. Our planner will select a combination of filtering methods $\{F_1, \ldots, F_n\}$ to apply on tracks in each connected component C, and corresponding pruning thresholds $\{T_1, \ldots, T_n\}$. The optimal parameters are query-dependent, so the planner chooses the best methods and thresholds for each query. Then, during query execution, we prune a candidate track A if the prediction of any filtering method falls below the corresponding threshold, i.e., $F_i(A) < T_i$. (Planning may also select zero filters, in which case no candidates are pruned.) Filtering outputs a set of candidates $filter(C)$ containing each track A such that $(F_1(A) > T_1) \wedge \ldots \wedge (F_n(A) > T_n)$. If $filter(C)$ is empty, we stop processing C and return to tracking. Otherwise, we pass $filter(C)$ to uncertainty resolution and refinement.

4.2.1 Nearest Neighbor Distance Filter. Intuitively, tracks that satisfy P should be more similar to each other than tracks that don't satisfy P. The Nearest Neighbor Distance (NND) filter builds on this intuition by leveraging a path distance function, i.e., a distance that can be computed between two tracks. Let D be the chosen distance function.

We use $P(S_{\text{train}})$, the set of tracks computed in training segments satisfying P, as example tracks that satisfy the predicate. Then, NND predicts whether a coarse track A produced during query execution satisfies P based on the distance from A to its nearest neighbor in $P(S_{\text{train}})$. Formally, $NND(A) = \min_{B \in P(S_{\text{train}})} D(A, B)$. Thus, we expect that $NND(A)$ is lower for tracks that satisfy P and higher for tracks that do not satisfy P. We use $-NND(A)$ as the filter prediction.

We implement the track distance function D with an approximation for the discrete Frechet distance [7] suitable for comparing a coarse track $A = \langle a_1, \ldots, a_n \rangle$ with a fine-grained track $B = \langle b_1, \ldots, b_n \rangle$. We first find the closest detection b_{i_1} to a_1, and compute the Euclidean distance d_1 from a_1 to b_{i_1}. Then, for a_2, we find the closest detection b_{i_2} such that $i_2 \geq i_1$, and compute the distance d_2. We compute distances for the remaining detections in A similarly, and compute $D(A, B)$ as the average of $d_1, \ldots, d_n$. This distance function accounts for the direction of the track (the distance between a track and its reversal will be high), is efficient to compute, and does not make assumptions about the position of the coarse track A between the sampled detections.

4.2.2 RNN Filter. The RNN filter applies machine learning to remain effective across a wider range of predicate types. Rather than make assumptions about the predicate, as we did in the NND filter for shape-based predicates, we train a lightweight recurrent neural network (RNN) to determine whether a track satisfies the predicate given the bounding boxes associated with detections along the track.

We use an RNN model consisting of an LSTM cell with 32 hidden state nodes. The RNN inputs a sequence of features from a coarse track: for each detection in the track, it inputs the 2D position, width, and height of the bounding box. We pass the last output of the RNN through an additional fully connected layer and sigmoid activation, and train the RNN with cross entropy loss, where tracks that do not satisfy P are labeled 0 while tracks that satisfy P are labeled 1.

We use S_{train} as training data for the RNN. Tracks $B \in S_{\text{train}}$ are accurate and fine-grained since they were computed during pre-processing by tracking objects at the full video framerate. During query execution, though, filtering will evaluate coarse tracks where detections are f frames apart. Thus, to effectively train the RNN on tracks similar to those we will see during execution, we define a function coarsify(B) that produces from B a coarse track by randomly pruning detections from B. Specifically, given a track $B = \langle d_{j_1}^{k_1}, \ldots, d_{j_n}^{k_n} \rangle$, we select a random integer $r \in [0, f)$, and produce a subsequence track B' containing each detection $d_{j_i}^{k_i}$ in B only if $k_i \bmod r = 0$. Then, for each $B \in S_{\text{train}}$, we derive a training example where the RNN input is coarsify(B) and the label is $P(B)$, which may not equal $P(\text{coarsify}(B))$.

During query execution, the RNN filter prediction is the probability output by the RNN given a candidate track A.

4.3 Uncertainty Resolution

Rather than perform uncertainty resolution after filtering, a naive approach is to process the intermediate video frame $I_{k+f/2}$ immediately after observing a matching output graph between I_k and I_{k+f} with nondeterministic edges. Processing $I_{k+f/2}$ may allow us to eliminate nondeterministic edges since the GNN should be more robust when matching detections between video frames that are only $f/2$ frames apart. However, oftentimes, uncertain matching decisions have no impact on the query outputs. For example, if a query seeks cars that rapidly decelerate (hard braking), then we do not need to address uncertainty in a group of nondeterministic tracks that all go through a traffic junction without slowing. Thus, processing additional video frames each time an uncertain matching decision is encountered wastes resources, especially for selective queries.

Instead, we lazily resolve uncertainty: we process a group C of candidate tracks sharing uncertain matching decisions only after each track in C has terminated, and only after we have pruned tracks that we are confident do not satisfy the predicate during filtering. Then, we only apply uncertainty resolution on groups C where $filter(C)$ is non-empty.

Uncertainty resolution processes C by deriving from $filter(C)$ a set of deterministic tracks $resolve(C)$ that correspond to correct matching decisions. We resolve all uncertain matching decisions that impact at least one nondeterministic track in $filter(C)$ by processing additional video around the uncertain decision at higher framerates.

Uncertainty resolution considers each vertex d_i^l that falls along some track in $filter(C)$. If d_i^l has multiple incoming edges, we address the uncertainty by applying the object detection model on $I_{l-f/2}$, and associating detections using the CNN+GNN model between $(I_{l-f}, I_{l-f/2})$ and between $(I_{l-f/2}, I_l)$. We then update C to reflect the matching graphs output by the model. First, we eliminate edges in C that connect detections between I_{l-f} and I_l. We then insert into C any edges from the bipartite matching graphs that are incident on a detection d_i^{l-f} or d_i^l in C. We perform a similar process for vertices d_i^l with multiple outgoing edges.

Recall that nondeterministic edges arose at a vertex v because the probability along multiple edges incident on v exceeded $Q_f \cdot MMP(v)$ for $Q_f < 1$. At the doubled framerate processed above, where sampled video frames are only $f/2$ frames apart, we may set $Q_{f/2} = 1$ during planning so that all edges are guaranteed to be deterministic. But, if $Q_{f/2} < 1$, then we may need to recursively process additional video frames at even higher framerates since the matching model may have low confidence (according to the parameter $Q_{f/2}$)

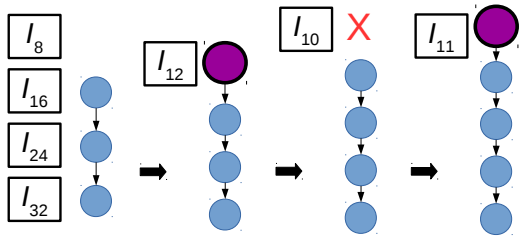

Figure 6: Prefix-Suffix refinement performs binary search to identify the first and last detections pertaining to a track. Here, we show the prefix search process.

when matching $(I_{l-f}, I_{l-f/2})$ or $(I_{l-f/2}, I_l)$. This recursive procedure is guaranteed to terminate since $Q_1 = 1$, as we cannot sample frames faster than the video capture rate.

Once the procedure completes, since there are no nondeterministic edges remaining, C is transformed into a set of connected components, where each component is a deterministic track. We compute $resolve(C)$ by re-applying filtering methods over these tracks.

4.4 Refinement

Refinement collects detections at a finer temporal granularity along tracks in $resolve(C)$ until we are confident that P will evaluate correctly over the refined tracks. In general, there are two factors that may cause P to evaluate incorrectly over a coarse track. First, P may depend on missing portions of the coarse track before its first detection and after its last detection. In this situation, we must examine video frames preceding and succeeding the coarse track to extend the track in both directions. Second, when evaluating a predicate on a coarse track, we perform linear interpolation between successive detections along the track. If the actual track deviates substantially from the interpolated detections, we must examine additional video frames in between the sampled detections to mitigate the interpolation error.

Thus, we develop refinement methods to target each of these situations. Given a coarse track, these methods select additional video frames for processing. As with filtering, our planner will select an optimal combination of refinement methods for a particular query. Then, on each selected frame I_k, we run the object detector to obtain detections D^k, and then use the GNN tracking model to associate new detections with coarse tracks in $resolve(C)$. Finally, we evaluate P over each of the refined tracks. If P evaluates true, we include the track in the query outputs.

4.4.1 Prefix-Suffix. This method targets the first situation discussed above, where we must extend a coarse track with additional prefix and suffix detections. We identify the first and last detections associated with the track using a binary search procedure, which we illustrate in Figure 6.

Prefix-Suffix refinement uses a single parameter, a minimum sampling frequency f_{min}, selected during planning. f_{min} constrains the level at which we stop the binary search. During evaluation, we only consider frame I_k if $k \mod f_{\text{min}} = 0$.

4.4.2 Acceleration. This method targets the second situation, where intermediate detections derived through linear interpolation have high error. Intuitively, linear interpolation likely yields high error along segments of the track where the object has high acceleration. Thus, for each triplet of consecutive detections (d_{i-1}, d_i, d_{i+1}), we compute the acceleration $accel(i) = (d_{i+1} - d_i) - (d_i - d_{i-1})$. We refine high-acceleration segments of the track — if $accel(i)$ exceeds a threshold T_{accel}, we examine additional video frames to capture additional detections between d_{i-1} and d_i, and between d_i and d_{i+1}, until the maximum acceleration between any pair of consecutive detections along the track is less than T_{accel}.

4.4.3 RNN Methods. As in filtering, we develop RNN-based refinement methods corresponding to the Prefix-Suffix and Acceleration methods. When we have enough tracks in the pre-processed segments for training the RNNs, and when there are consistent patterns in when refinement is needed, these methods can reduce the overall refinement cost by selecting video frames in a more targeted manner.

RNN-Prefix-Suffix trains an RNN to predict whether a coarse track requires additional prefix and suffix detections. We provide the RNN with the same inputs as in Section 4.2.2, but train it to output two probabilities, one for the prefix and one for the suffix. If these predictions are lower than a threshold, we do not explore the prefix or suffix.

RNN-Interp-Error trains an RNN to estimate the maximum error between linearly interpolated detections and the correct positions on each segment of the coarse track. Unlike our other RNN models, here we use an RNN that outputs an error prediction on each step. We refine by exploring additional intermediate video frames along each segment where the RNN error prediction exceeds a threshold.

5 PLANNING

Given a query, our query planner selects the minimal sampling framerate at which we initially execute object tracking, along with a combination of filtering methods, the Q_f parameters that control when we consider a matching output graph to contain uncertainty, and a combination of refinement methods. The planning objective is to select parameters that minimize the number of video frames over which we must apply the object detector while satisfying user-specified accuracy guarantees. Specifically, users provide an accuracy bound α (e.g., $\alpha = 99\%$) requiring that the accuracy of fast

query-driven tracking be within α percent of the accuracy of tracking at full-framerate.

Our planner considers exponentially increasing minimal sampling framerates, and greedily selects filtering, uncertainty resolution, and refinement parameters for each framerate. Then, we execute the query using the sampling framerate and corresponding parameters that yield the highest execution speed while providing accuracy above α. Throughout our planner, we use the training and validation segment tracks S_{train} and S_{val} computed during pre-processing to evaluate the performance of different parameter choices.

Below, we first introduce our approach to select filtering, uncertainty resolution, and refinement parameters for a particular sampling framerate. We then detail the overall planning process that considers and evaluates plans at exponentially increasing framerates.

5.1 Filtering

Planning for filtering involves selecting a set of filter methods $\{F_1, \ldots, F_n\}$, along with pruning thresholds for each method $\{T_1, \ldots, T_n\}$, for a given sampling frequency f. During query execution, we prune a track A if its prediction under any of the selected methods falls below the corresponding threshold, i.e., $F_i(A) < T_i$. We may also select zero methods, in which case we perform no candidate pruning.

We evaluate selections of methods and thresholds based on their precision and recall over the validation tracks S_{val}. Let coarsify(S_{val}) be a set derived from S_{val} by randomly coarsifying each track (defined in Section 4.2.2). Then, for a particular combination of methods $\{F_1, \ldots, F_n\}$, let $S_{\text{filter}} = \{B \mid B \in \text{coarsify}(S_{\text{val}}), F_1(B) > T_1 \wedge \ldots \wedge F_n(B) > T_n\}$ be the subset of coarsified validation tracks that pass filtering. We define filtering precision as $\frac{|S_{\text{filter}} \cap S_{\text{val}}|}{|S_{\text{filter}}|}$ and recall as $\frac{|S_{\text{filter}} \cap S_{\text{val}}|}{|S_{\text{val}}|}$.

The search objective is to find the methods and thresholds that yield highest filtering precision with recall at least α. For a given set of methods $\{F_1, \ldots, F_n\}$, we select thresholds using a simple brute force algorithm. For each method, we select $m = 100$ candidate thresholds that evenly divide the method's predictions for tracks in coarsify(S_{val}). We then enumerate each of the m^n threshold combinations, and select the combination with highest precision and recall at least α. We repeat the threshold search for each combination of filter methods. Although the execution time is exponential in the number of methods, in practice we only implement two filter methods, and we find that the search completes in only 4 sec with three methods and 1,000 tracks since filter predictions can be pre-computed and cached.

5.2 Uncertainty Resolution

We now describe how to set the parameters $Q_f, Q_{f/2}, \ldots, Q_1$ used during uncertainty resolution. At a high level, we set these thresholds sufficiently small so that α percent of tracks in S_{val} known to satisfy P are tracked correctly.

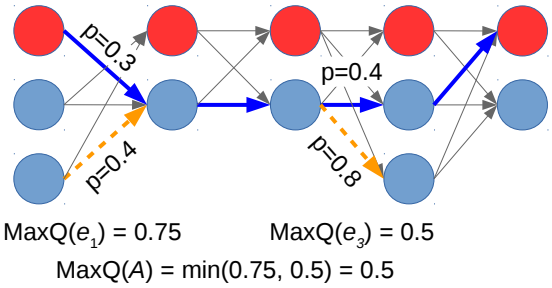

Figure 7: Uncertainty planning to select the Q_f parameter at some sampling frequency. Here, we compute $MaxQ$ for one track. Blue, bold edges show the path of the track through the probability graph, while orange, dashed edges represent cases where the GNN assigns a higher probability to an incorrect edge.

We first execute GNN matching over the pre-processed segments of video to derive an association graph H at each sampling frequency f. Rather than pruning low-confidence edges in H as we would during query execution, we assume $Q_f = 1$ during tracking so that no edges are pruned; additionally, we annotate each edge in H with the GNN output probability. Then, for each track $B \in P(S_{\text{val}})$ known to satisfy the predicate, we follow the edges corresponding to the track through H, and compute the fraction $MaxQ(e) = \frac{p^k_{i,j}}{MMP(d^k_i)}$ for each edge $e = (d^k_i, d^{k+f}_j)$ along B. $MaxQ(e)$ is the largest setting of Q_f that would correctly retain the edge during the tracking process; for most edges, $MaxQ(e) = 1$ since the GNN model usually outputs the maximum probability along correct edges. We define $MaxQ(B)$ to be the minimum over the $MaxQ(e)$ values of edges along the track. We show an example of computing $MaxQ(B)$ in Figure 7.

Finally, we set Q_f to the α-th percentile $MaxQ(B)$ value over all of the tracks in $P(S_{\text{val}})$. This ensures that α percent of object instances that appear in the query outputs are tracked correctly during query execution.

5.3 Refinement

During planning we select up to two refinement methods, one for capturing prefix and suffix detections (Prefix-Suffix or RNN-Prefix-Suffix) and one for capturing intermediate detections (Acceleration or RNN-Interp-Error). Each method uses one or more thresholds to decide when to terminate refinement.

We compute thresholds for one or two refinement methods using an algorithm similar to Section 5.1. After determining thresholds for each combination of refinement methods, we

select the combination that yields query accuracy at least α while requiring processing of the fewest additional video frames. Filtering and uncertainty threshold parameters are decided prior to refinement planning. Thus, we compute the number of additional video frames that a combination of refinement methods examines by applying filtering, uncertainty resolution, and refinement over the validation video.

5.4 Planning Algorithm

Our overall planning algorithm initially sets the maximum sampling frequency $f = 1$, and iteratively doubles f. For each value of f, we select filtering, uncertainty resolution, and refinement parameters using the approaches discussed above. We pick the setting of f and corresponding execution plan that requires sampling the fewest video frames during tracking, uncertainty resolution, and refinement over the validation segments. For repeated queries over the same dataset, we account for frames over which we already computed object detections that can be reused.

To minimize planning time, we never consider frequencies f that exceed half the length of the shortest track in $S_{\text{train}} \cup S_{\text{val}}$ — such sampling frequencies always yield ineffective query plans, since tracking would either miss tracks entirely or only capture one detection for some tracks.

Planning is fast because we use cached object detection outputs over the pre-processed video segments to evaluate all enumerated execution plans.

6 JOIN QUERIES

Above, we have detailed our approach for queries that select individual tracks. Join queries that select multiple tracks may include both predicates $P_1, \ldots, P_n$ that evaluate individual tracks and a join predicate P_{join} evaluating a tuple of tracks $A_1, \ldots, A_n$. When processing such queries, we begin by independently computing the association graph through tracking for each output column A_i in the query. This yields sequences of groups of nondeterministic tracks, $S_1 = \langle C_{1,1}, C_{1,2}, \ldots, \rangle, S_2 = \langle C_{2,1}, \ldots \rangle, \ldots$. We first apply filtering independently in each sequence S_i for the individual track predicates P_i. Then, we select tuples $(C_{1,i_1}, \ldots C_{2,i_2}, \ldots)$ of temporally overlapping groups of tracks among the sequences, with $C_{j,i_j} \in S_j$, and apply filtering for P_{join} on each tuple; here, we use simple extensions of our NND and RNN filters for multi-track predicates. Finally, we apply uncertainty resolution and refinement as before on individual components C_{j,i_j} that appear in at least one tuple.

7 EVALUATION

7.1 Dataset

We built a diverse dataset of video data, object track queries, and hand-labeled query outputs to evaluate MIRIS. Our dataset includes five sources of video data, which we denote as BDD, UAV, Tokyo, Warsaw, and Resort. These sources are:

- BDD — the Berkeley DeepDrive dataset [22], consisting of 1,100 hours of video from dashboard cameras on motor vehicles.
- UAV — two hours of video we captured from a UAV hovering above a traffic junction.
- Tokyo, Warsaw — 60 hours of video from fixed cameras at traffic junctions in Tokyo and Warsaw, respectively.
- Resort — 60 hours of video from a fixed camera pointed towards a pedestrian walkway outside of a resort hotel.

We manually labeled bounding boxes around objects in one to four hundred video frames in each data source (except BDD, since BDD includes labels), and trained several YOLOv3 [18] object detection models to input video at variable resolutions and output detections. We detect cars in UAV, Tokyo, and Warsaw, and pedestrians in BDD and Resort, at 30% confidence threshold, which we empirically find yields the best overall query accuracy.

We formulate five distinct queries over object tracks that each pertain to a subset of the video sources:

Turning movement count (Q1, Q2, Q3). In the UAV (Q1), Tokyo (Q2), and Warsaw (Q3) videos, we identify cars that traverse the traffic junction along a particular turning direction. These queries apply predicates of the same form, requiring the first detection in a track to be contained in a region of the camera frame corresponding to a particular source road, and the last detection to fall on a particular destination road.

Pedestrians crossing in front (Q4). Q4 identifies tracks of pedestrians in BDD who cross directly in front of the vehicle on which the camera is mounted, from one side of the vehicle to the other (left to right or right to left). This query may be executed by a data analyst investigating anomalous automonous vehicle behavior in certain situations.

Car stopped in crosswalk (Q5). In the Tokyo video, we observe several instances where a vehicle stops in the middle of a crosswalk, either because the traffic light just turned red or to pick-up or drop-off passengers. Q5 selects car tracks that stop for fifteen or more seconds on a crosswalk.

Joggers (Q6). Q6 identifies joggers in the Resort video by selecting tracks with average speed exceeding a threshold. We empirically set this threshold to 15 pixel/sec by comparing tracks of two joggers against tracks of walking pedestrians. Variations of this query could be useful to distinguish the frequencies of different activities in various urban locations.

Hard braking (Q7). A traffic planner interested in improving traffic safety may wish to identify instances of hard braking around a junction. Q7 selects car tracks from Warsaw that decelerate from 8 m/s to less than 1 m/s within three seconds.

Turn on red (Q8). Traffic planners may also be interested in identifying instances where cars turn on a red light for safety analysis. Q8 selects car tracks from the Warsaw dataset that turn right on red. The traffic light is not visible in the video, so we determine the light state based on cars that go straight through the junction — we use a join query that selects triplets of car tracks ($A_{\text{turn}}, A_{\text{pred}}, A_{\text{succ}}$) such that A_{turn} turns right from west to south, while A_{pred} and A_{succ} go straight through the junction from north to south, and A_{pred} precedes A_{turn} while A_{succ} succeeds A_{turn}.

Straight on red (Q9). Similar to Q8, Q9 selects cars from Tokyo that go straight through the junction on a red light at least four seconds before and after the light turned green. In this dataset, the traffic light is visible, so we simply select tuples ($A_{\text{turn}}, A_{\text{red}}$), where A_{turn} is a car that travels south-to-north through the junction, and A_{red} is the track of a red traffic light that controls traffic along the north-south direction.

7.2 Baselines

We compare MIRIS against four baselines: Overlap, PP, KCF, and FlowNet. Overlap implements a standard multi-object tracking algorithm, IOU, that computes object tracks by associating object detections between frames based on overlap [3, 4]. This approach is used in video query systems such as BlazeIt [12] and Rekall [9], and achieves state-of-the-art accuracy when object track annotations are not available [4].

PP implements the Deep Neural Network (DNN) classifier in probabilistic predicates [16]. We train the DNN to classify whether 5-second segments of video contain any object instances that satisfy the query predicate. During query execution, we skip segments of video that the DNN has high confidence contain no relevant object instances. The PP baseline is also similar to specialized NNs in BlazeIt [12] and approximate filters in SVQ [21].

KCF applies kernelized correlation filters [11], an unsupervised tracking algorithm that tracks an object across video given its position in an initial frame. KCF is extremely efficient as it does not require object detections, but exhibits lower accuracy than tracking-by-detection approaches like Overlap. In our implementation, we apply the object detector on key frames at a variable frequency (e.g., once per second), and use KCF to track objects through intermediate frames.

FlowNets [5] are convolutional neural networks trained to estimate optical flow (displacement of objects between frames) that have been successfully applied in fast object detection approaches such as Deep Feature Flow [23] to update detection bounding boxes across sequences of frames. This baseline applies a FlowNet to associate detections between two consecutive frames based on the optical flow output.

7.3 Metrics

For queries Q1-Q8, we hand-label a 50-minute segment of video with the expected query outputs. To measure tracking accuracy, we compare the tracks produced by an object tracking approach with the hand-labeled tracks in terms of precision and recall. Let $match$ be the number of correct tracks, fp be the number of output tracks that do not appear in the hand-labeled set (false positives), and fn be the number of hand-labeled tracks that were missed. Then, precision and recall are defined as:

$$\text{precision} = \frac{match}{match + fp} \quad \text{recall} = \frac{match}{match + fn}$$

We measure accuracy in terms of a single F1 score computed as the harmonic mean of precision and recall. We match output tracks with our hand-labeled tracks based on counts in each two-minute segment of video: if there are c output tracks in one two-minute segment, and c^* is our hand-labeled count for the segment, then the segment contributes $\min(c, c^*)$ matches, $\max(0, c - c^*)$ false positives, and $\max(0, c^* - c)$ false negatives.

7.4 Results

We first evaluate the approaches in terms of the speed-accuracy tradeoff that they provide for Q1-Q8 over the 50-minute video segments where we have hand-labeled query outputs. We measure speed on an NVIDIA Tesla V100 GPU in terms of the video-to-processing-time ratio, i.e., the number of video seconds that each approach processes in one GPU-second. Here, we focus on expensive GPU operations since CPU-bound operations in MIRIS and the baselines can be heavily optimized and parallelized. Additionally, we consider only query execution time and exclude pre-processing and planning time; our approach targets queries over large datasets with thousands of hours of video, and phases in MIRIS prior to query execution would be negligible on such datasets. We perform a separate experiment in Section 7.6 with Q9 where we consider the end-to-end query processing runtimes, including video decoding, pre-processing, and planning.

Figure 8 shows results for the four baselines, our GNN tracking algorithm without filtering, uncertainty resolution, and refinement (denoted GNN), and our full query-driven tracking approach (denoted MIRIS). We show curves over varying sampling framerates: 10 fps, 5 fps, 2.5 fps, 1.25 fps, 0.8 fps, 0.6 fps, 0.4 fps, 0.3 fps, and 0.2 fps. For MIRIS, we also vary the query-driver tracking parameters corresponding to different settings of the accuracy bound α. Points higher and to the right represent better performing algorithms. For each query, we fix the input video resolution by using the resolution that yields the highest accuracy for Overlap at 10

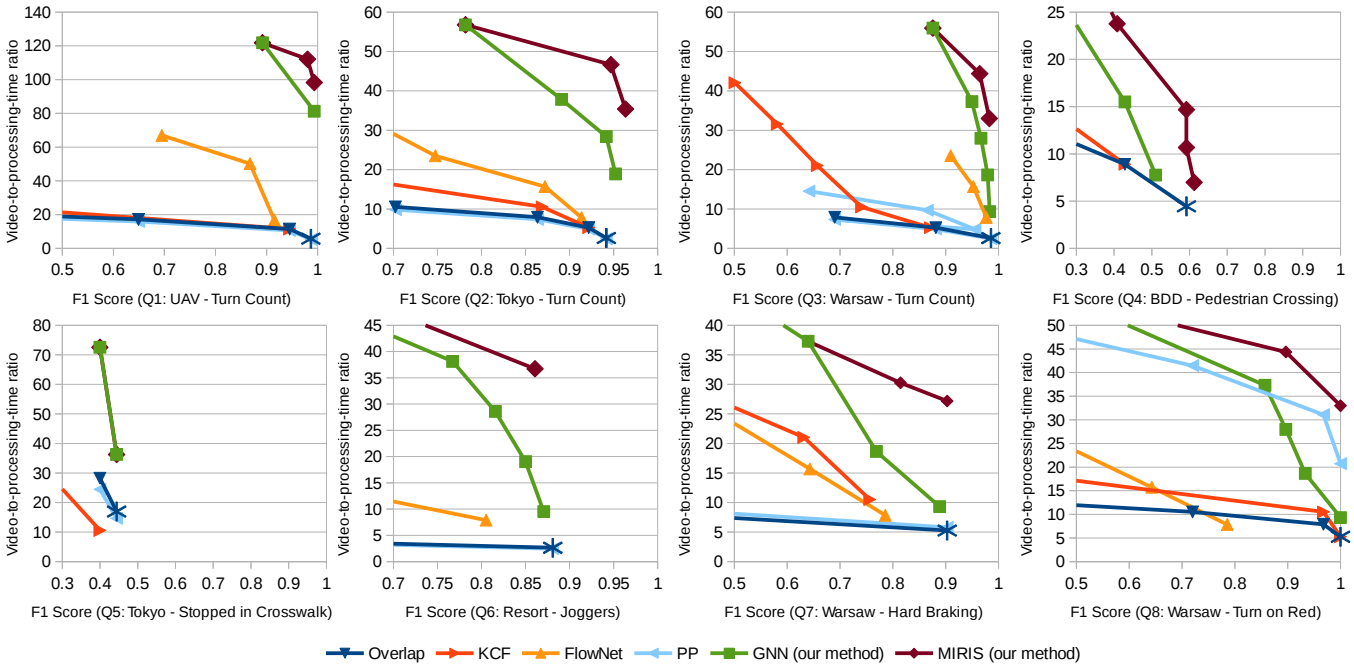

Figure 8: Speed-accuracy curves for MIRIS and the four baseline approaches over Q1-Q8.

fps. We test the following resolutions: 1920×1080, 1460×820, 960 × 540, 736 × 414, 640 × 360, and 576 × 324.

We highlight the accuracy of tracks computed by Overlap at the full video framerate with asterisks. This accuracy, which compares output tracks against hand-labeled tracks, is poor for certain queries (especially Q4 and Q5) because Overlap makes errors when it encounters intersecting objects and other challenging scenarios. Since we train our approach on tracks computed by Overlap in the pre-processed segments, this accuracy also represents a soft bound on the maximum accuracy of GNN and MIRIS, along with the other baselines.

Even without query-driven tracking, our GNN low-framerate tracking approach provides a substantial speedup over the other baselines on almost every query. Whereas Overlap, KCF, and FlowNet suffer reduced accuracy as the sampling framerate is reduced below 5-10 fps, GNN tracking generally remains accurate at 1-2 fps.

Still, at sampling framerates below 1 fps, GNN tracking accuracy suffers due to an increasing number of incorrect matching decisions, and increasingly coarse tracks over which the query predicates do not correctly evaluate. Thus, for every query but Q5, MIRIS provides an additional speedup through query-driven tracking: by pruning tracks that MIRIS is confident do not satisfy the query, and then applying uncertainty resolution and refinement on the remaining tracks, the system is able to further accelerate query execution. The speedup of MIRIS over GNN is particularly high for selective queries such as Q6-Q8.

PP performs comparably to Overlap for most queries because the DNN filter is unable to eliminate a significant number of video frames. In Q1-Q3, the DNN filter fails because the object instances selected by the query appear in virtually every video frame. In Q4-Q7, although relatively few segments of video contain relevant object instances, the DNN filter is ineffective because the differences between relevant frames and irrelevant frames are subtle, and there is not enough training data for the neural network to learn these subtle patterns. For instance, in Q6, joggers occupy a very small portion of a busy pedestrian walkway, and in order to differentiate frames containing joggers, the DNN would need to learn to track pedestrian speeds over its sequence of input frames and threshold the maximum speed. PP performs well in Q8 by skipping frames where the density of cars indicates that the traffic light is green.

KCF and FlowNet generally do not perform well, and on several queries, they do not appear in the chart because their highest accuracy is low and far to the left. We find that KCF frequently fails to accurately track objects due to partial occlusion, varying object sizes, and lighting changes. The FlowNet model performs well in some queries (Q1-Q3, Q6), but even for these queries, it yields a lower speed-accuracy tradeoff than GNN because it learns to track objects in an indirect way (through optical flow).

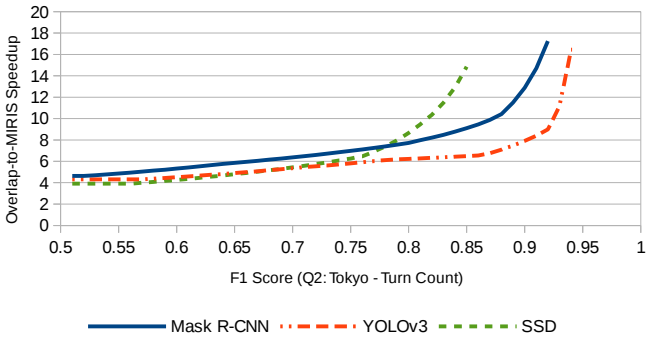

Figure 9: Speedup of MIRIS over Overlap on Q2 with YOLO, Mask R-CNN, and SSD object detectors.

Table 2: End-to-end performance of Overlap, GNN, and MIRIS on Q9 over 60 hours of video in the Tokyo dataset. We estimate dollar cost assuming execution over 72K hours of video.

Method	Pre-proc	Plan	Exec	Cost	F1
Overlap	0 min	0 min	2106 min	$129K	100%
GNN	0 min	0 min	175 min	$11K	67%
MIRIS	58 min	11 min	220 min	$13K	100%

7.5 Object Detection Algorithms

Other work on video query systems [12, 23] often employ Mask R-CNN [10] or SSD [15] instead of YOLO. We show that MIRIS offers substantial cost savings across different object detectors by comparing the speedup of MIRIS over Overlap on Q2 with varying detectors in Figure 9. We use TensorFlow implementations of Mask R-CNN and SSD [1, 2], and the Darknet YOLO library [19]. We configure YOLOv3 and Mask R-CNN to input 960×540 images, whereas SSD inputs resized 512 × 512 frames due to limitations in the available network model. MIRIS consistently provides a 12-14x speedup over Overlap at the highest achievable accuracy.

7.6 End-to-end Performance

In Section 7.4, we focused on the execution time of GPU-intensive operations. Here, we evaluate the end-to-end performance of Overlap, GNN, and MIRIS on executing Q9 over the full 60-hour Tokyo dataset, including pre-processing, planning, and video decoding. We use a 48-thread machine with one NVIDIA Tesla V100 GPU. We fix the video resolution at 960 × 540 and, for Overlap, we fix the sampling framerate at 10 fps. Since hand-labeling query outputs over the entire dataset is too time-consuming, we use tracks computed by Overlap as "ground truth". Note that in reality the methods make mistakes, as shown in Figure 8. However, we do verify that the 6 tracks identified by Overlap for Q9 are correct instances where cars pass the junction on a red light; we show one example in Figure 10.

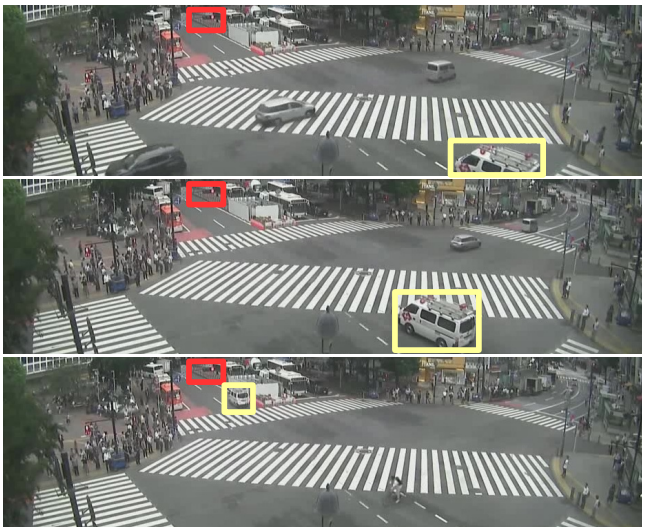

Figure 10: Example car and red light that satisfy Q9.

We show the results in Table 2. MIRIS executes the query with maximum sampling frequency $f = 16$ (0.6 fps), and we use this parameter for GNN as well. Although GNN reduces query execution time 12-fold without requiring pre-processing or planning, it only achieves an F1 score of 67%. On the other hand, including pre-processing and planning time, MIRIS yields 100% accuracy with a 7x speedup. In MIRIS, RNN training dominates planning time. MIRIS execution includes 95 minutes for video decoding, 85 minutes for object detection, 10 minutes for GNN inference, and 30 minutes for filtering, refinement, and uncertainty resolution (primarily capturing additional object detections). Training times for YOLO (24 hr) and the GNN model (4 hr), which are performed only once per video source, are not included.

We also show the estimated dollar cost to execute Q9 over a hypothetical larger 72K-hour video dataset on AWS. This dataset size corresponds to video captured from 100 cameras over one month. On large datasets, pre-processing and planning costs are negligible. In this scenario, MIRIS achieves a massive 10× cost reduction from $129K to $13K.

8 CONCLUSION

MIRIS accelerates the execution of object track queries by 9x on average by integrating query processing into the object tracker. By resolving uncertainty and refining tracks with finer-grained detections only when needed for a query, MIRIS reduces the number of video frames that must be processed. MIRIS broadens the types of queries that can be optimized by video query processors, which are becoming crucial as video is captured in ever-increasing volumes.

REFERENCES

[1] Waleed Abdulla. 2017. Mask R-CNN on Keras and TensorFlow. https://github.com/matterport/Mask_RCNN.

[2] Paul Balanca. 2017. Single Shot MultiBox Detector in TensorFlow. https://github.com/balancap/SSD-Tensorflow.

[3] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. 2016. Simple Online and Realtime Tracking. In *IEEE International Conference on Image Processing (ICIP)*. IEEE, 3464–3468.

[4] Erik Bochinski, Tobias Senst, and Thomas Sikora. 2018. Extending IOU Based Multi-Object Tracking by Visual Information. In *IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 1–6.

[5] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. 2015. FlowNet: Learning Optical Flow with Convolutional Networks. In *IEEE International Conference on Computer Vision (ICCV)*. 2758–2766.

[6] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2224–2232.

[7] Thomas Eiter and Heikki Mannila. 1994. *Computing Discrete Fréchet Distance.* Technical Report. TU Wien.

[8] Lex Fridman, Daniel E Brown, Michael Glazer, William Angell, Spencer Dodd, Benedikt Jenik, Jack Terwilliger, Aleksandr Patsekin, Julia Kindelsberger, Li Ding, et al. 2019. MIT Advanced Vehicle Technology Study: Large-Scale Naturalistic Driving Study of Driver Behavior and Interaction with Automation. *IEEE Access* 7 (2019), 102021–102038.

[9] Daniel Fu, Will Crichton, James Hong, Xinwei Yao, Haotian Zhang, Anh Truong, Avanika Narayan, Maneesh Agrawala, Christopher Re, and Kayvon Fatahalian. 2019. *Rekall: Specifying Video Events using Compositions of Spatiotemporal Labels.* Technical Report. Stanford University.

[10] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask R-CNN. In *IEEE International Conference on Computer Vision (ICCV)*. 2961–2969.

[11] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. 2014. High-Speed Tracking with Kernelized Correlation Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37, 3 (2014), 583–596.

[12] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. Challenges and Opportunities in DNN-Based Video Analytics: A Demonstration of the BlazeIt Video Query Engine. In *Conference on Innovative Data Systems Research (CIDR)*.

[13] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Neural Network Queries over Video at Scale. In *Proceedings of the VLDB Endowment*.

[14] Sanjay Krishnan, Adam Dziedzic, and Aaron J Elmore. 2019. DeepLens: Towards a Visual Data Management System. In *Conference on Innovative Data Systems Research (CIDR)*.

[15] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. SSD: Single Shot MultiBox Detector. In *European Conference on Computer Vision (ECCV)*. Springer, 21–37.

[16] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. 2018. Accelerating Machine Learning Inference with Probabilistic Predicates. In *International Conference on Management of Data (SIGMOD)*. ACM, 1493–1508.

[17] Alan Lukezic, Tomas Vojir, Luka Cehovin Zajc, Jiri Matas, and Matej Kristan. 2017. Discriminative Correlation Filter with Channel and Spatial Reliability. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6309–6318.

[18] Joseph Redmon and Ali Farhadi. 2018. *YOLOv3: An Incremental Improvement.* Technical Report. University of Washington.

[19] Joseph Chet Redmon. 2015. Darknet CNN Library. https://github.com/pjreddie/darknet.

[20] Mohammad Shokrolah Shirazi and Brendan Tran Morris. 2016. Vision-Based Turning Movement Monitoring: Count, Speed & Waiting Time Estimation. *IEEE Intelligent Transportation Systems Magazine* 8, 1 (2016), 23–34.

[21] Ioannis Xarchakos and Nick Koudas. 2019. SVQ: Streaming Video Queries. In *International Conference on Management of Data (SIGMOD)*. ACM.

[22] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. 2017. End-to-end Learning of Driving Models from Large-scale Video Datasets. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2174–2182.

[23] Xizhou Zhu, Yuwen Xiong, Jifeng Dai, Lu Yuan, and Yichen Wei. 2017. Deep Feature Flow for Video Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2349–2358.