arXiv:2507.09448v1 [cs.DB] 13 Jul 2025

TRACER: Efficient Object Re-Identification in Networked Cameras through Adaptive Query Processing

Pramod Chunduri
Georgia Institute of Technology
pramodc@gatech.edu

Yao Lu
National University of Singapore
luyao@comp.nus.edu.sg

Joy Arulraj
Georgia Institute of Technology
arulraj@gatech.edu

***Abstract*—Efficiently re-identifying and tracking objects across a network of cameras is crucial for applications like traffic surveillance. Spatula is the state-of-the-art video database management system (VDBMS) for processing RE-ID queries. However, it suffers from two limitations. Its spatio-temporal filtering scheme has limited accuracy on large camera networks due to localized camera history. It is not suitable for critical video analytics applications that require high recall due to lack of support for adaptive query processing.**

In this paper, we present TRACER, a novel VDBMS for efficiently processing RE-ID queries using an adaptive query processing framework. TRACER selects the optimal camera to process at each time step by training a recurrent network to model long-term historical correlations. To accelerate queries under a high recall constraint, TRACER incorporates a probabilistic adaptive search model that processes camera feeds in incremental search windows and dynamically updates the sampling probabilities using an exploration-exploitation strategy. To address the paucity of benchmarks for the RE-ID task due to privacy concerns, we present a novel synthetic benchmark for generating multi-camera RE-ID datasets based on real-world traffic distribution. Our evaluation shows that TRACER outperforms the state-of-the-art cross-camera analytics system by 3.9× on average across diverse datasets.

***Index Terms*—video database management systems, object re-identification, adaptive query processing**

I. INTRODUCTION

Advances in computer vision over the last decade have led to increased interest in the development of Video Database Management Systems (VDBMSs) that automatically analyze videos at scale [1]–[5]. These systems primarily focus on – (1) *object queries* - retrieving general instances of objects categories (*e.g.*, cars, animals) [1], [2], [6], (2) *action queries* - retrieving action segments (*e.g.*, left turn of a car, pedestrian crossing from left to right) [7]–[9], (3) *object track queries* - tracking a specific object within a single camera [10], [11].

In this paper, we focus on another important computer vision task – *re-identifying* and tracking an object across a network of cameras [12], [13], as shown in Figure 1. This *object re-identification* query has many real-world applications. For instance, a traffic monitoring expert might be interested in retrieving the trajectory of a suspicious vehicle from extensive video footage captured within a city environment. An insurance professional may seek to retrospectively examine particular vehicle trajectories to verify instances of insurance fraud.

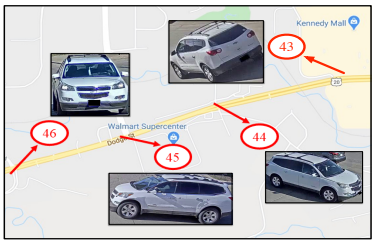


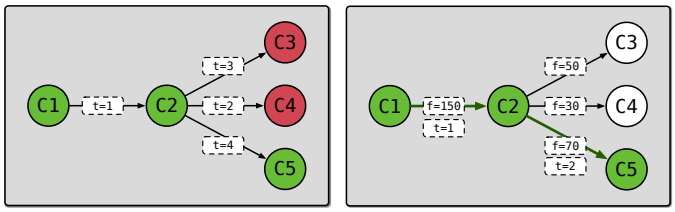
Fig. 1: Object Re-identification over a Camera Network.

RE-IDENTIFICATION TASK. The task of multi-camera object re-identification (RE-ID) requires identifying all instances of an object across a camera network using a single query image of the object [13]. A NAÏVE pipeline for processing a RE-ID *query* involves running - (1) an object detector (*e.g.*, YOLOv5 [14]) on individual frames of all the cameras, (2) a re-identification model (*e.g.*, fine-grained ResNet backbones [15]) on all the detected objects to extract fine-grained features, and (3) similarity matching with the source image feature vector.

As we are only interested in *spotting* the object in each camera, the VDBMS terminates the search within a camera once the object is found. The high computational cost of object detection and subsequent feature extraction makes it impractical to implement this pipeline across the entire camera network. The pipeline runs at a low frame rate of 1 fps on a CPU and 10 fps on a server-grade GPU.

Modern VDBMSs propose proxy model-based filtering techniques to quickly filter out frames that do not contain the object category of interest (*e.g.*, car) [1], [2], [6]. However, these techniques only work well when the video dataset exhibits a low occupancy rate [2]. City-scale traffic scenarios rarely exhibit this behavior due to high traffic density, resulting in a limited performance gain.

BASELINE #1: GRAPH SEARCH. The camera network can be represented as a graph wherein cameras are depicted as nodes, and edges connect cameras that are directly accessible from one another, such as through adjacent roads. Given a query object with a source camera and a time stamp, the RE-ID query reduces to finding the path traversed by the object through the *camera graph*. As shown in Figure 2a, a baseline algorithm for finding the object trajectory consists of iteratively traversing the neighboring cameras of each camera wherein the object is



(a) GRAPH-SEARCH traverses the camera network in random order.

(b) SPATULA uses spatial correlations to predict the most probable camera.

Fig. 2: Baseline approaches for processing RE-ID queries.

spotted. Similar to canonical graph traversal algorithms, this algorithm explores the neighboring cameras in random order. We refer to this algorithm as GRAPH-SEARCH.

BASELINE #2: SPATULA. Spatula [16] is the state-of-the-art (SoTA) system for cross-camera video analytics. As shown in Figure 2b, Spatula uses spatio-temporal filtering on top of the camera graph to process RE-ID queries. More specifically, SPATULA uses *localized camera history* at each camera to predict the order of the most probable neighboring cameras and the timestamp to contain the object. In the example shown in Figure 2b, SPATULA predicts C5 as the next camera from C2 since 70 objects in the past traversed from C2 to C5 while only 50 and 30 objects traversed to C3 and C4, respectively. This way, SPATULA generates a dense spatial cross-camera correlation matrix to accelerate RE-ID queries. While SPATULA improves upon GRAPH-SEARCH by using a localized camera history, both techniques suffer from three limitations that constrain their runtime performance in practice.

CHALLENGES. Efficiently processing RE-ID queries requires solving three main challenges:

1. Large Search Space. The first challenge is that the search space for the query object in a RE-ID query is quite large. A 24-hour video dataset obtained from a city camera network, comprising 1000 cameras capturing footage at 10 frames per second, consists of approximately 0.9 billion frames. Thus, the goal is to considerably reduce the number of frames to examine for each RE-ID query. GRAPH-SEARCH reduces the search space by leveraging the underlying graph structure of the camera network and avoiding irrelevant cameras (*e.g.*, far apart cameras). However, the random exploration strategy employed in GRAPH-SEARCH still results in a significant computational cost when the camera network exhibits high connectivity (degree), as the system may need to process multiple camera feeds before locating the correct camera. SPATULA reduces the search space by predicting a ranked ordering of the most probable (next camera, next frame) pair that contains the object using a localized camera history. However, the spatio-temporal filtering model in SPATULA is not always accurate as it relies on localized camera history to compute correlations between adjacent cameras. This often results in the processing of incorrect neighboring cameras and start frames.

2. Recall-oriented processing. The second challenge is that the target applications of RE-ID queries often require a high recall. Consider a traffic monitoring expert tracking an Amber alert. Nearly all cameras containing the suspect vehicle must be retrieved. To achieve high recall, the system needs to exhaustively process the frames in a camera until the object is found. If the system examines a camera that does not contain the object, it has to process the entire camera feed before moving to the next camera. Both GRAPH-SEARCH and SPATULA do not take an *adaptive query processing* approach for selecting the neighboring cameras. So, the low camera prediction accuracy of these systems leads to the processing of several additional frames in each camera before moving to the next camera.

3. Lack of large-scale video datasets for re-id task. The third challenge is the lack of large-scale multi-camera video datasets to evaluate RE-ID queries. The largest existing real-world dataset for RE-ID [13] suffers from two limitations: (1) The dataset is characterized by a small number of cameras (<20) in a confined geographical area, with each camera capturing less than 5 minutes of video footage. (2) The dataset lacks the necessary camera history information essential for performing camera prediction tasks in systems like SPATULA. Collecting large-scale synchronized multi-camera video datasets for RE-ID poses challenges due to the need for access to real-world camera networks. Further, the dataset must undergo rigorous checks to ensure that user privacy is preserved [17].

OUR APPROACH. In this paper, we present TRACER, a VDBMS tailored for efficiently processing RE-ID queries. TRACER operates on the underlying camera graph using a novel adaptive query processing framework. Specifically, TRACER performs a probabilistic adaptive search over the graph network in small temporal windows using an exploration-exploitation strategy. TRACER utilizes the camera prediction probabilities at each camera to *sample* the most probable neighboring camera at each time step while exploring a small temporal window of pre-defined duration. Based on the outcomes of the current window, TRACER dynamically updates the probabilities for the subsequent sampling iteration. This way, TRACER performs a dynamic incremental search over all the neighboring cameras to spot the object of interest efficiently.

Note that the sampling probabilities play a crucial role in determining the adaptive search process. For example, a highly skewed probability array would result in a rigorous exploitation of the highest probable camera before exploring other cameras. So, a highly accurate camera prediction model would result in often examining the correct cameras rigorously before examining any other cameras. We observe that the most probable next camera at any given camera is dependent on the long-term trajectory of the object to reach this camera and not just the local camera trajectory that is used in SPATULA. The intuition behind this observation is that real-world object trajectories often contain long-term dependencies (*e.g.*, popular routes in a neighborhood). So, TRACER learns the long-term cross-camera correlations to predict the most probable next cameras. Specifically, TRACER trains a recurrent network on the historical trajectory information to learn the most probable next camera, given a sequence of cameras traversed so far. Thus, the probabilistic search model combined with an accurate camera

	Multi Camera Tracking	Spatial Filtering	Temporal Filtering	Adaptive Processing	Long-term Correlations
Proxy Models [1], [2], [6]			✓		
MIRIS/OTIF [10], [11]			✓		✓
GRAPH-SEARCH	✓	✓			
SPATULA [16]	✓	✓	✓		
TRACER	✓	✓	✓	✓	✓

TABLE I: Systems for processing RE-ID queries. TRACER (1) tracks object across cameras, (2) performs spatio-temporal filtering, (3) adaptively processes cameras in the network, and (4) leverages long-term cross-camera correlations to accelerate the query.

prediction model allows TRACER to process RE-ID queries efficiently. Table I shows a qualitative comparison of TRACER against existing systems for processing RE-ID queries.

To evaluate TRACER, we present a new large-scale synthetic benchmark for multi-camera RE-ID, based on the Carla simulator [18]. This benchmark allows spawning an arbitrary number of cameras in arbitrary locations and orientations in a game engine. We develop a technique to simulate real-world traffic scenarios by mimicking real-world trajectory distributions. The benchmark allows the simulation of different traffic patterns (*e.g.*, traffic density, distribution skew) to facilitate further research in VDBMSs.

CONTRIBUTIONS. In summary, the key contributions are:

- We highlight the limitations of SoTA spatio-temporal filtering systems in processing RE-ID queries over a large camera graph (§I). We present a novel adaptive query processing system to overcome these limitations (§III).
- We model long-term cross-camera correlations using a recurrent network to improve the camera prediction accuracy (§V).
- We develop a probabilistic adaptive search algorithm that processes camera feeds in small temporal windows and dynamically updates the probabilities using an exploration-exploitation strategy (§VI).
- We develop a novel RE-ID benchmark capable of generating multi-camera video datasets with diverse data distributions (§VII).
- We evaluate TRACER on synthetic benchmarks and city-scale real-world GPS trajectory datasets. We show that TRACER accelerates RE-ID queries by 4.7× on average compared to GRAPH-SEARCH and by 3.9× on average compared to SPATULA (§VIII).

ETHICS STATEMENT. In this paper, we explore efficient multi-camera object re-identification, which has both positive and negative applications. On the positive side, this technology benefits urban planning, emergency protocols, sports analytics, and more. However, there are potential risks, including privacy concerns with fine-grained traffic surveillance [19], [20]. To mitigate these risks, we emphasize responsible and ethical use of the technology [21]. We have taken concrete steps, including releasing an open-source synthetic benchmark for privacy-preserving dataset generation and using anonymized datasets in our analysis. By emphasizing transparency and privacy protection, we strive to contribute to the responsible development of object re-identification technology.

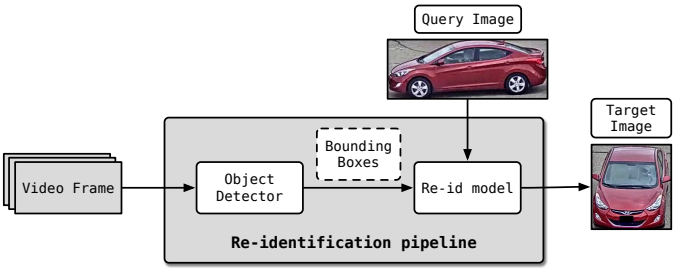


Fig. 3: A typical pipeline for object re-identification.

II. BACKGROUND

OBJECT RE-IDENTIFICATION. Object re-identification (RE-ID) is the task of retrieving all the instances (target images) of an object from a supplied query image. When applied to multi-camera video feeds, RE-ID enables tracking objects over a network of cameras, enabling multi-camera multi-target tracking (MCMT) [13]. For example, a traffic monitoring expert may be interested in finding the route taken by a suspicious vehicle in the city-scale road network. RE-ID is challenging because of: (1) significant variations in the scene, distance, and viewing angles of different cameras and (2) occlusion due to other objects and camera placements.

RE-ID PIPELINE. Figure 3 shows the typical object re-identification pipeline to retrieve the target image given a video frame and a query image. An object detector first processes the video frame to find all the objects, and returns the object bounding boxes. The bounding boxes and the query image are processed by a RE-ID *model* to extract their respective feature vectors [15]. Subsequently, the similarity between each object's feature vector and the query image's feature vector is computed using the standard Cosine or Euclidean distance to identify the best matches.

RE-ID MODELS. The RE-ID *model* determines the accuracy of the RE-ID pipeline above in finding correct matches. Two main classes of models have been used for RE-ID in literature.

1. Feature engineering. Early RE-ID models [22] used hand-crafted techniques like histogram of oriented gradients (HOG) [23] to generate feature vectors based on edge- and keypoint- detection. However, these hand-crafted local features do not generalize to scene/view changes.

2. Deep neural networks. Recent advancements in RE-ID involve deep neural networks (DNNs) to improve generalization. Part-based models extract features for semantic partitions of the objects (*e.g.*, hood, trunk of a vehicle) and aggregate these part-based features. Global feature-based models, on the other hand, use global object features and incorporate rigorous data augmentation to account for scene, orientation, and size changes. The SoTA technique [15] for detecting fine-grained object features uses a global feature-based model with three large DNN backbones (variants of ResNet [24]) to generate a merged feature vector. So, feature extraction for RE-ID is computationally expensive, and up to 2× slower than standard object detection models such as YOLOv5 [14].

CHALLENGES. In the city-scale vehicle RE-ID task, time-synchronized videos are ingested from several cameras [13]. The objective is to find all the occurrences of the object in all the cameras given a query image. This task has applications in various domains, such as traffic monitoring, urban planning, and sports analytics. However, finding all the occurrences requires: (1) detecting objects in each frame in all cameras, (2) performing RE-ID feature extraction for all detected objects, and (3) finding pairwise similarity (*e.g.*, cosine) between the source object and each detected object. Given the vast number of detected objects, this task incurs significant computational costs. For example, with 16 cameras capturing 5-minute sequences at 10 fps, querying a single object requires 72,000 RE-ID pipeline invocations, resulting in a total computation time of about 2 hours on a server-grade GPU.

RE-ID QUERIES. In many tasks, it is not necessary to find *all the occurrences* of the object in each camera. For instance, in time-sensitive applications like Amber alert tracking, the primary objective is to *spot the suspect vehicle* in the cameras (*i.e.*, we need one timestamp per camera to track the vehicle in the camera network). However, achieving a high recall rate is essential for the query to ensure exhaustive identification, meaning that no cameras can be missed in the process. We formulate this task as a RE-ID *query*: given a query image, the source camera, and the time-stamp of occurrence, find all the cameras in the network that contain the query image. To process such RE-ID queries efficiently, our goal is to identify the query object in all the cameras that contain it using the fewest number of RE-ID pipeline invocations.

III. SYSTEM OVERVIEW

TRACER assumes that a camera network topology is given as an unweighted graph $G = (V, E)$. In this graph, each vertex $v \in V$ represents a camera and each edge $e \in E$ represents a connection between two adjacent cameras. Note that adjacent cameras in the graph might be physically far away from each other (*e.g.*, highway exits). We assume that each camera captures videos of length T frames synchronously at a fixed frame rate. We assume that the video feeds from all the cameras are recorded and sent to a central server. These video feeds are updated periodically.

The input to TRACER is a query object bounding box q, a source camera $c_q \in V$, and the time stamp $t_q \in [1, T]$ when the query object was found in c_q. TRACER's goal is to find the occurrences of the query object in the camera network rapidly. Specifically, TRACER returns the target object bounding box q' along with the associated camera and timestamp pair $< c'_q, t'_q >$, one for each camera that contains the query object. Note that we are only interested in spotting the vehicle in each camera feed. So, we only need to find one frame in each camera that contains the object.

Figure 4 shows the system architecture of TRACER. As discussed in the problem setup, TRACER takes a query image q with the source camera and time stamp $< c_q, t_q >$ as input. It operates on a synchronous multi-camera video feed of length T frames collected from the camera graph G. TRACER consists of two novel components that contribute to its high performance.

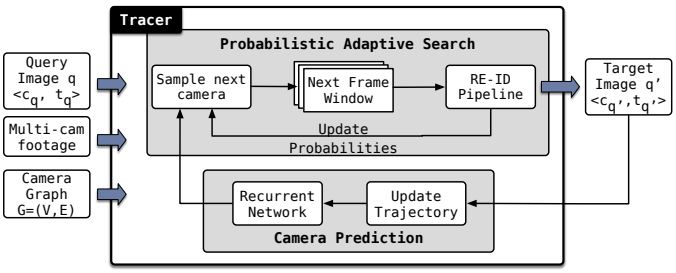


Fig. 4: System architecture of TRACER.

PROBABILISTIC ADAPTIVE SEARCH. TRACER employs a probabilistic adaptive search strategy to achieve high recall efficiently. As shown in Figure 4, TRACER dynamically selects the next camera to process at each time step from candidate cameras within the graph. The selection involves sampling from a probability distribution generated by the camera prediction module (§V). Once a camera is chosen, TRACER incrementally processes a window of its frames, executing the RE-ID pipeline (Figure 3). If the query object is detected, TRACER returns the target image q', camera c'_q, and timestamp t'_q. Additionally, TRACER also appends c'_q to the current trajectory and uses it as input to the camera prediction module. However, if the query object is not found, TRACER updates the probabilities using an exploration-exploitation rule (§VI) and proceeds to the next sampling round. This allows TRACER to retrieve all relevant cameras by only incrementally exploring video feeds.

CAMERA PREDICTION. TRACER features a camera prediction module that accurately predicts the next camera to examine at each time step (§V). The module's primary output is the probability of finding the object in each of the neighboring cameras. On dense camera networks with a high degree, the choices for the next camera are large. In such cases, we find that local cross-camera correlations (Figure 2b) are insufficient for accurate camera prediction. To address this issue, the camera prediction module in TRACER trains a recurrent neural network (§V-D). The trained network takes the trajectory traversed thus far as input and outputs probabilities for finding the object in each neighboring camera. By utilizing long-term trajectories, TRACER effectively captures cross-camera correlations even between distant cameras within the camera graph, enhancing the accuracy of camera prediction.

Both the camera prediction and the probabilistic adaptive search modules operate in conjunction to enhance the performance of TRACER. Specifically, the notable enhancement in accuracy provided by the camera prediction module (VIII-D) allows TRACER to frequently select the correct camera with confidence. In such scenarios, the search module facilitates effective exploitation of the identified camera to quickly identify the target object without unnecessary delays caused by incorrect cameras. In the event that the camera prediction module produces inaccurate results, the search module promptly adjusts to a random exploration strategy to prevent being trapped

by incorrect cameras. TRACER thus efficiently executes the RE-ID query over the camera network using a combination of probabilistic adaptive search and accurate camera prediction.

IV. PROBLEM FORMULATION

We begin by formally defining the problem statement. As discussed in §III, we are given a camera network $G = (V, E)$ and a query image q with source camera, timestamp pair $< c_q, t_q >$. The naïve approach of brute-force searching through all the cameras in the network becomes infeasible due to the large number of cameras and the substantial number of frames within each camera. The goal is to output all the cameras that contain the query object in the camera network G while examining as few frames as possible. The search space reduction problem can be formally defined as follows:

Problem 1 (Search Space Reduction): Given a camera network $G = (V, E)$, a query object bounding box q, along with a source camera and timestamp pair $< c_q, t_q >$, the objective is to *minimize the number of frames examined* to retrieve all the cameras $V' \subseteq V$ that contain the query object.

Leveraging our knowledge of the underlying graph structure, we can limit the search space to the reachable cameras from c_q and employ a graph traversal technique (GRAPH-SEARCH) to locate the target object. GRAPH-SEARCH traverses the camera network G from the source camera c_q and iteratively explores the neighboring cameras until it locates the target object (Figure 2a). However, this approach requires examining all the frames within each camera before moving to the next camera. Additionally, it requires examining several additional cameras since it explores the neighboring cameras in random order. The oracle solution to this problem (ORACLE), which has access to the ground truth object cameras and timestamps, can retrieve the subset of cameras V' by examining $|V'|$ frames (one frame per camera). The ORACLE can be visualized as a traversal algorithm that always selects the correct neighboring camera to explore and always finds the object in the first frame that it examines. This is the best possible solution to the search space reduction problem and serves as the upper bound for any solution. A close-to-optimal solution to Problem 1 should examine only a small subset of frames within each camera and predict the neighboring camera order with high accuracy.

PROBLEM SCOPE. Objects often appear in multiple consecutive frames in a camera due to temporal continuity. However, recall that our primary objective is only to spot the object in each camera. Once we detect the object in a camera, we output the target bounding box q', along with the corresponding camera and timestamp tuple $< c'_q, t'_q >$, and proceed to search for the object in the next camera.

Additionally, objects may appear multiple times in the same camera. Our system natively supports detecting objects reappearing in the same camera after several camera hops (*e.g.*, vehicle loops). However, rapid oscillations of objects between two cameras lead to poor recurrent model convergence. Sparse camera networks at city scale rarely exhibit such trajectory patterns [13]. So, TRACER currently does not support it.

V. CAMERA PREDICTION

We first describe the camera prediction module of TRACER. Let us define the event of finding the query object q in the camera v as v_q. At each camera u, our goal is to estimate the probability of finding the query object q in the set of neighboring cameras $v \in V_n$, written as $P(v_q)$. To maintain brevity, we will omit the object q from the notation throughout the remainder of this paper and utilize $P(v)$ in place of $P(v_q)$.

A. Local correlations

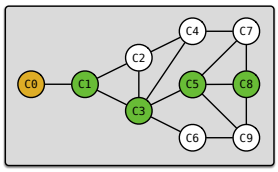
To estimate this probability, SPATULA computes the Maximum Likelihood Estimate (MLE) of the probability using the historical frequency count of objects that traveled from u to each of the neighboring cameras $v \in V_n$.

$$P_{MLE}(v) = \frac{C(v)}{N}$$

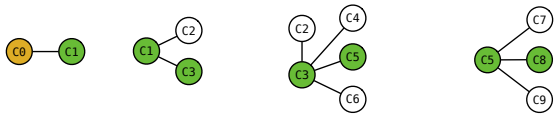
where $C(v)$ is the frequency count of camera v in the historical object tracks and $N = \sum_{u \in V_n} C(u)$ is the total count of all the objects in all the neighboring cameras. We can compute $P_{MLE}(v)$ for all the neighboring cameras and generate a probability array over the neighboring cameras. Thus, this approach generates a search order for the neighboring cameras by utilizing the localized historical information to identify the neighboring cameras that have been visited most frequently. However, the drawback of this approach is that a simplistic frequency estimate that only looks at the localized camera information lacks the necessary granularity to estimate the probability accurately. Our experiments show that this approach achieves less than 50% accuracy in predicting the neighboring camera order even on sparse camera networks (§VIII-D).

B. Long-term Correlations

To improve the accuracy of camera prediction, we propose a novel approach that utilizes the long-term correlations between cameras to accurately estimate the probability of finding the query object in the neighboring cameras. For example, consider a scenario where two neighboring cameras, A and B, have been visited an equal number of times in the past. In this case, relying solely on the frequency estimate would treat both cameras as equally probable, even though the historical object trajectories may indicate that the object is more likely to be found in camera A. Our intuition for this approach is that object trajectories in the real world are likely to follow certain path distributions. For example, in a city, there are likely to be specific traffic routes that are more popular than others due to ease of access. Furthermore, cities typically comprise hotspots for both the source and destination points, which leads to certain trajectories being more probable than others. For example, an analysis of the New York City taxi dataset [25] reveals that 80% of the trips originate or terminate at 10% of the total locations (Figure 9). More formally, let us denote the path taken by the object to reach the current camera u_{k-1} as $\{u_1, u_2, \ldots, u_{k-1}\}$, where $k-1$ is the path length so far from the source vertex u_1. Then, our objective is to estimate the probability of finding the object O in neighboring camera



(a) Illustrative camera network topology – A query image found in camera C0 at time step t=0 is used to track the object across the network.



Time Step	t=1	t=2	t=3	t=4
GRAPH-SEARCH search order	C1	C2, C3*	C4, C2, C5*, C6	C9, C7, C8*
SPATULA input	C0	C1	C3	C5
SPATULA search order	C1	C3*, C2	C6, C2, C4, C5*	C9, C8*, C7
TRACER input	[C0]	[C0, C1]	[C0, C1, C3]	[C0, C1, C3, C5]
TRACER search order	C1	C3*, C2	C6, C5*, C2, C4	C8*, C7, C9
Total cameras processed				
GRAPH-SEARCH/SPATULA/TRACER	1/1/**1**	2/1/**1**	3/4/**2**	3/2/**1**

(b) Camera prediction in TRACER on the given graph – TRACER efficiently processes RE-ID queries by accurately predicting the order of processing the neighboring cameras.

Fig. 5: An illustrative example of TRACER's camera prediction module

u_k of u_{k-1} conditioned on the path taken so far, written as $P(u_k|u_1, u_2, \ldots, u_{k-1})$.

C. N-gram approach

An obvious choice to estimate this conditional probability is the n-gram model [26]. N-gram models are a popular technique used in natural language processing to estimate the probability of a word given the previous n-1 words. Consider the object trajectory in the camera graph as a sequence $\overline{u}$ of k cameras from a starting camera u_1. Using the Markov assumption, n-gram models simplify the probability estimation to use a fixed number of previous cameras to predict the next camera. For a given n, the probability is defined as:

$$\begin{aligned} P(\overline{u}) = P(u_k|u_{k-1}, u_{k-2}, \ldots, u_{k-n+1}) \\ \times P(u_{k-1}|u_{k-2}, u_{k-3}, \ldots, u_{k-n}) \\ \times \ldots \\ \times P(u_2|u_1) \end{aligned} \tag{1}$$

For n=1 (*unigram*), this probability presumes the strong independence condition and reduces to the simple frequency estimate used in SPATULA (§V-A):

$$P(\overline{u}) = \prod_{i=1}^{k} P(u_i)$$

To train general n-gram models, we must compute the probability $P' = P(u_k|u_{k-1}, u_{k-2}, \ldots, u_{k-n+1})$ in Equation (1) using all the available sequences of length n in the dataset. More specifically, for each neighbor u_k of camera u_{k-1}, we need to find the MLE of P'. For each neighbor u_k, this probability can be estimated as:

$$\begin{aligned} P'_{MLE}(u_k) &= \frac{P(u_{k-n+1}, \ldots, u_{k-1}, u_k)}{P(u_{k-n+1}, \ldots, u_{k-1})} \\ &= \frac{Count(u_{k-n+1}, \ldots, u_{k-1}, u_k)}{Count(u_{k-n+1}, \ldots, u_{k-1})} \end{aligned}$$

In other words, for each neighboring camera, we must compute the frequency of occurrence of the sequence $< u_{k-n+1}, \ldots, u_{k-1}, u_k >$ and the sequence $< u_{k-n+1}, \ldots, u_{k-1} >$. To compute these quantities for any given sequence, we use the historical data of the vehicle trajectories. We traverse each trajectory in a sliding window fashion to update the counts for each n length sequence. The first $n-1$ cameras in the sequence are used as the index to update the frequency count for the n^{th} camera. At inference time, we use the last n cameras in the current trajectory of the object to index into the n-gram model and generate the list of probabilities for the neighboring cameras.

D. Recurrent Network

The use of n-gram models to capture long-term spatial correlations is suboptimal due to the lack of prior knowledge on the optimal length of n. Moreover, the optimal length of n might vary across different segments of the trajectory. For example, consider a vehicle trajectory that starts at a highway and ends at a city. The optimal length of n may vary between the highway and city segments of the trajectory, as the traffic distribution in these two segments is different. Consequently, our experiments show that the n-gram model improves the prediction accuracy by only 8% on average over the frequency estimate used in SPATULA (§VIII-D).

To address this issue, we propose the use of recurrent neural networks (RNN) to capture the long-term spatial correlations. RNNs are a class of neural networks that are tailored for sequential data. They use feedback connections to pass information from one time step to the next [27]. This allows them to capture long-term dependencies in the data. RNNs preserve information across time steps by maintaining a hidden state h_t that serves as a proxy representation of the past input sequences. The ability of RNNs to handle variable-length sequences makes them highly suitable for our problem. The RNN model improves the prediction accuracy by 25% on average over SPATULA (§VIII-D).

ILLUSTRATIVE EXAMPLE. Figure 5 shows an example of TRACER's camera prediction pipeline. Consider the 10 camera topology in Figure 5a. Given a query image (*e.g.*, vehicle bounding box) captured in camera C0 at time $t = 0$, Figure 5b shows the processing states of GRAPH-SEARCH, SPATULA, and TRACER at each time step. At $t = 1$, all systems pick C1 as the next camera, as it is C0's only neighbor. At $t = 2$, C2 and C3 are C1's neighbors, with the object in C3. GRAPH-SEARCH randomly predicts the search order C2, C3. SPATULA

uses the current camera C1 to predict the search order C3, C2. TRACER uses the current trajectory [C0, C1] as input to the RNN and predicts the search order C3, C2. Since the object is in C3, GRAPH-SEARCH processes one additional camera, while SPATULA and TRACER process no additional cameras.

At $t = 3$, GRAPH-SEARCH, SPATULA, and TRACER explore 2, 3, and 1 additional cameras, respectively. Both GRAPH-SEARCH and SPATULA are inaccurate in this step - GRAPH-SEARCH due to the random prediction order, and SPATULA because it only considers the localized camera history at C3. In contrast, TRACER leverages the full trajectory [C0, C1, C3] to predict a more accurate order. Note that TRACER also incurs additional cost here, albeit lower than the other baselines.

Finally, at $t = 4$, GRAPH-SEARCH and SPATULA explore 2 and 1 additional cameras, respectively. TRACER's RNN correctly predicts the next camera using the current trajectory [C0, C1, C3, C5]. Despite mispredicting the camera in the previous time step, the RNN's current input remains the ground-truth trajectory due to the 100% recall constraint. After 4 time steps, TRACER selectively processes 5 of 10 cameras, while GRAPH-SEARCH and SPATULA process 9 and 8 cameras.

TRAINING THE RNN MODEL. We use an LSTM network [28] with one hidden layer (128 units) as the RNN model. For training, a batch of sequences are fed as the input, while the corresponding sequences, right-shifted by 1, are used as labels. The RNN is thus trained to predict the next camera in the sequence effectively, regardless of the sequence length. A fully-connected layer on the final hidden state produces the neighboring camera probability distribution. The model is trained using the Adam optimizer [29] with a learning rate 0.001. During inference, the trained RNN model iteratively predicts the next camera by incrementally extending the input trajectory length, as illustrated in Figure 5b.

TRAINING DATA ACQUISITION. For real-world deployment, the ground truth trajectories for training can be generated offline in a time-insensitive manner by running the RE-ID pipeline on historical video feeds. Additionally, the inference trajectories from recent queries can be used as ground-truth training trajectories for subsequent queries. This allows TRACER to periodically update the prediction model and readily handle traffic distribution drift [30]. We plan to explore more efficient ways of handling distribution drift in future work.

VI. PROBABILISTIC ADAPTIVE SEARCH

With the RNN model, we can predict the next camera in the sequence with high accuracy. However, the RNN model does not achieve 100% accuracy, and it might mispredict the order of the camera search. For example, consider the scenario in Figure 5b. The RNN model mispredicts the camera search order at time step 3, and starts the search at camera C6 instead of C5. An exhaustive search model searches all the frames in a predicted camera, and so this misprediction would result in processing all the frames in C6 before searching C5.

To address this concern, we present a probabilistic adaptive search model that analyzes camera frames in small temporal windows, guided by the probability distribution predicted by the RNN model. More precisely, the next camera is sampled from the neighboring cameras using the probability distribution predicted by the RNN model. Subsequently, we search within a fixed window of frames in the sampled camera. The window size is tuned once per camera network based on the average duration of the objects within a camera view. In case the object is not detected in the sampled camera, we update the probability distribution using an exploration-exploitation strategy that balances the exploration of new cameras with the exploitation of the predicted camera.

PROBABILITY UPDATE ALGORITHM. Let the initial probability array be $P = [p_1, p_2, \ldots, p_n]$, where p_i is the probability for the i^{th} neighboring camera. After exploring a camera i in a sampling round, the updated probability array $P' = [p'_1, p'_2, \ldots, p'_n]$ can be calculated as follows:

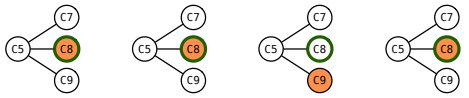
$$p'_i = \alpha \cdot p_i$$

$$p'_j = p_j + \frac{p_i * (1 - \alpha)}{n - 1} \quad \text{for } j \neq i$$

In this update algorithm, p'_c represents the updated probability for each neighboring camera c, α is the exploration factor, and n is the total number of neighboring cameras. This update algorithm can be interpreted as a way to balance the exploration of new cameras with the exploitation of the predicted most probable camera. The exploration factor α serves as a hyperparameter that can be tuned to achieve the desired balance between exploration and exploitation. A higher value of α results in a slower reduction of the probability of the most probable camera, and thus, a higher exploitation rate.

The intuition behind this update algorithm is that a highly accurate RNN model often produces a skewed probability distribution. When the underlying learned model is highly accurate, the value of α is set close to 1 to preserve the original probability distribution to a certain degree. When the object is present in the most probable camera, the rigorous exploitation strategy enables TRACER to locate the object within the initial sampling rounds, during which the probability of the most probable camera remains high. However, if the object is not found in the most probable camera, the exponential reduction in the probability of the most probable camera leads to a more balanced probability distribution within a few subsequent sampling rounds. In cases of less accurate models (*e.g.*, SPATULA), α can be set to a lower value to promote increased exploration of neighboring cameras.

ILLUSTRATIVE EXAMPLE. Figure 6 shows an illustrative example of the probabilistic adaptive search model of TRACER. Specifically, we focus on the time step $t = 4$ in Figure 5b, where the current camera is C5 and the object is located in C8. To enhance clarity of presentation, assume that the search process is initiated from frame 1 and the object is located in the frame interval (180, 220) in C8. The search window is fixed at 75 frames, and the probabilities are updated after each sampling round using the update algorithm discussed above. At $t = 4$ in Figure 5b, the input to the recurrent network is the trajectory [C0, C1, C3, C5]. Utilizing this trajectory,



Sampling round	1	2	3	4
Probabilities	[0.1, 0.8, 0.1]	[0.2, 0.6, 0.2]	[0.3, 0.4, 0.3]	[0.4, 0.5, 0.1]
Sampled camera	C8	C8	C9	C8
Search window	(1, 75)	(76, 150)	(1, 75)	(151, 225)
Total Frames	75	150	225	255

Fig. 6: Probabilistic execution model in TRACER– TRACER dynamically updates the sampling probabilities for each neighboring camera and adaptively expands the search in each neighboring camera feed to process RE-ID queries efficiently. The object is eventually located in camera C8 during the frame interval (180, 220).

the recurrent network produces the probability array [0.1, 0.8, 0.1] for the neighboring cameras [C7, C8, C9], respectively. During the initial sampling round, the system selects camera C8 and searches within the window (1, 75). As the object is not detected in C8, the system updates the probabilities to [0.2, 0.6, 0.2]. In the second round, the system again samples camera C8 and performs a search within the window (76, 150) but fails to locate the object. The probabilities are now updated to [0.3, 0.4, 0.3].

In the third round, the system samples camera C9 since the probabilities are more uniform, and initiates the search in the window (1, 75). It is notable that the dynamic probability update algorithm enhances the chances of exploring neighboring cameras when the system fails to locate the object in the most probable ones. As the object is not detected, the probabilities are updated to [0.4, 0.5, 0.1], and the system selects camera C8 during the fourth round. Subsequently, the system identifies the object in frame 180 and outputs the target object's location and timestamp before transitioning to the next camera.

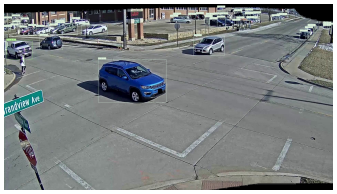
VII. MULTI-CAMERA RE-ID BENCHMARK

An essential challenge for optimizing and evaluating RE-ID queries is the lack of a large-scale multi-camera video dataset.

LIMITATIONS OF EXISTING DATASETS. The largest real-world multi-camera dataset is CityFlow from the Nvidia AI City Challenge [13]. CityFlow includes 3 hours of traffic videos from 40 cameras at 10 intersections in a US city. The videos are 1-5 minutes long, temporally unsynchronized, and hence unsuitable for temporal analysis. The most extensive map has under 10 intersections and 337 unique vehicles, severely limiting its usefulness to the RE-ID task. A key challenge in obtaining large real-world multi-camera RE-ID datasets is privacy concerns. Real-world datasets must be curated to remove personally identifiable information, such as license plates, before public release. For example, the Duke-MTMC dataset was released and then retracted due to privacy issues [17].

SYNTHETIC DATASETS. To circumvent these limitations of real-world datasets, researchers have recently proposed synthetic datasets such as VisualRoad [31] and Synthehicle [32]. However, they do not meet the following requirements for evaluating RE-ID queries:

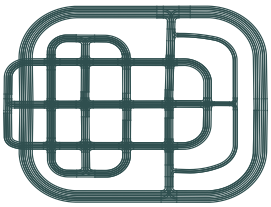
1. Long synchronized multi-camera video feeds. The dataset should comprise video feeds from multiple synchronized



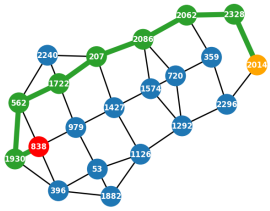
(a) Sample frame from the AI city dataset - The cameras are typically deployed at intersections.

(b) Sample frame generated using Carla - We calibrate the camera positions to mimic real videos.

Fig. 7: Qualitative comparison of real and simulated data



(a) Road network of the Town05 map in Carla - cameras are deployed at the intersections in the road network



(b) Camera graph generated automatically from the road network, with an annotated sample vehicle track

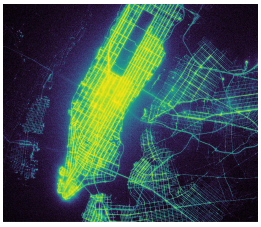
Fig. 8: Carla maps and corresponding camera networks

cameras, ensuring that vehicles adhere to temporal constraints, and the video feeds should be sufficiently long to capture multiple vehicle tracks across the network. VisualRoad is not suitable for the RE-ID task due to its inability to generate temporally synchronized multi-camera videos.

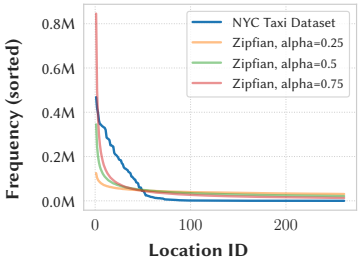
2. Realistic data. The simulated data must closely follow the real-world data distribution and behavior. Specifically, the dataset should contain (1) realistic vehicle movements, such as adherence to traffic rules (2) realistic traffic patterns, including traffic hotspots and long-tail distributions. However, VisualRoad and Synthehicle are not capable of generating such datasets.

CITY-SCALE VIDEO DATA GENERATION. We generate a large-scale city traffic video dataset using the Carla Simulator [18]. Carla is an open-source autonomous driving simulator built on top of Unreal Engine [33]. It provides features such as realistic driving agents, synchronized simulation, and the ability to spawn cameras at arbitrary locations. Figure 7 visually compares frames from the Cityflow and Carla datasets. Notice how we can calibrate the camera position to simulate real-world camera deployment. Additionally, these cameras can synchronously capture the world at a specified frame rate for an arbitrarily long time.

Carla provides multiple maps with a varying number of road networks and intersections. Figure 8a shows an example map provided by Carla. Using the Carla API, we can automatically spawn cameras at each intersection in the road network with minimal calibration. We then automatically generate the camera network graph as shown in Figure 8b to use as input to TRACER. Carla offers the capability to incorporate custom road networks and world elements, enabling the



(a) Heatmap of pickup locations in NYC Taxi dataset.



(b) The pickup/dropoff locations loosely follow a Zipfian distribution.

Fig. 9: Source/destination point distribution - We generate trajectory data by sampling the pickup and dropoff locations from a Zipf distribution with varying skew factors.

simulation of extensive camera networks that adhere to real-world geographical limitations.

Most importantly, Carla controls the spawning and driving behaviors of the vehicles. We use CARLA Agent scripts [34] to simulate individual vehicle driving behavior. Specifically, we set arbitrary spawn and destination points for each vehicle and then use local and global route planning algorithms to navigate from the spawn point to the destination point. The vehicles navigate via the shortest path from source to destination while interacting with other vehicles, following traffic lights, signs, and speed limits.

To simulate real-world traffic, we observe that city-scale traffic scenarios often exhibit hotspots, which are locations with a higher likelihood of being either the source or destination points. For example, a few areas downtown and a few residential neighborhoods in a city are more likely to be source and destination points for the vehicle tracks. We test this hypothesis on the real-world NYC Taxi dataset [25]. Figure 9a shows the heatmap of pickup locations for 10 million trips over a 1 month period. As seen in Figure 9b, the data loosely follows a Zipfian distribution with a few points accounting for most pickup locations. So, we use a Zipfian distribution with different skew factors to generate the traffic patterns in our dataset. Specifically, we sample the source and destination spawn points for each vehicle from the Zipfian distribution independently and navigate the vehicle through the map using the shortest path.

TRAINING DATA GENERATION. As discussed in §V-D, we train the recurrent network using historical ground truth trajectories from a given camera network. For the synthetic benchmark, the training data is generated by running the Carla simulator in a *no-rendering mode*. This mode produces ground truth traces 2× faster than visual data by simulating the whole environment without graphical rendering. Our experiments show that 2000 ground truth traces suffice to train a highly accurate camera prediction model for all queries with a similar distribution. The cost of generating 2K traces in the no-rendering mode is only 20× the per-query inference cost.

VIII. EVALUATION

In our evaluation, we illustrate that:

- TRACER executes RE-ID queries 3.9× faster on average than the SoTA system SPATULA, and 4.7× faster on average than GRAPH-SEARCH (§VIII-B).
- TRACER outperforms all the RE-ID baselines across various data skew levels and achieves close-to-ORACLE performance in highly skewed distributions (§VIII-C).
- TRACER's recurrent network-based camera prediction module achieves 25% higher accuracy than SPATULA's localized prediction module (§VIII-D), and the performance gap widens as the camera networks grow larger (§VIII-E).
- TRACER incurs negligible runtime overhead compared to expensive vision operators, and TRACER incurs a low per-query training cost (§VIII-F).

A. Experimental setup

EVALUATION SETTINGS. We evaluate TRACER on four network topologies from three datasets. Table II provides a summary of the key dataset statistics for the 4 topologies.

- **Carla-based synthetic scenarios:** We use the Carla-based synthetic scenarios generated using the benchmark described in §VII for the first two topologies.
- **PORTO:** For our third topology, we use the city of Porto, Portugal, and the Porto GPS trajectories dataset [35]. We designate 200 intersections in the Porto road network as cameras locations in the graph. The dataset contains 1.7 million GPS trajectories from 442 taxis over 12 months. We use a subset of 25,000 trajectories for training and evaluation. To simulate camera footage, we assume a camera deployed at each intersection captures a 200 square meter area at 10 fps. Since we lack access to the full vehicle occupancy at intersections, we assume the same average occupancy as the Cityflow dataset [13].
- **BEIJING:** For the fourth topology, we utilize the Geolife GPS trajectories dataset collected in Beijing, China [36]. This dataset includes 17,621 GPS trajectories annotated with 8 different transportation modes (walk, car, bus *e.t.c.*) over 5 years. We use trajectories from 4 modes - walk, car, taxi, and bus - for training and evaluation. We use the same methodology as the PORTO dataset to generate the camera graph and simulate the traffic footage.

BASELINES. We compare TRACER against five baselines.

- NAÏVE algorithm uses an object detector and a RE-ID model (Figure 3) on each frame of every camera to detect the presence of the query object, terminating the search in a camera once the object is found [37].
- PP leverages lightweight proxy models, as used in prior VDBMSs [1], [2], [6] to quickly filter irrelevant frames from the video. Specifically, we use PP to remove frames that do not contain any objects of the query class.
- GRAPH-SEARCH traverses the camera network starting from the source camera, randomly selecting and processing one neighboring camera at a time (Figure 2a).
- SPATULA [16] utilizes localized camera history (§V-A) to predict the most likely next camera and start frame, as shown in Figure 2b. Since we operate in the 100% recall

Topology	Data type	# Cameras	(Avg, Max) Degree	Total Duration	Avg. # objects per frame	Avg. Trajectory Length (# cameras)	# Training Trajectories
TOWN05	Synthetic Videos	21	(3.5, 4)	10.4 hrs	0.9	6.6	2298
TOWN07	Synthetic Videos	20	(3.2, 4)	16.0 hrs	1.4	8.2	2104
PORTO	GPS traces	200	(7.1, 8)	-	-	13.0	25000
BEIJING	GPS traces	200	(7.1, 8)	-	-	9.9	7091

TABLE II: Dataset Characteristics – We report these key characteristics of the datasets we evaluate TRACER on: (1) no.of cameras in the network, (2) avg. degree of the graph, (3) total no.of frames and avg. no.of objects per frame, and (4) average trajectory length and no.of training trajectories.

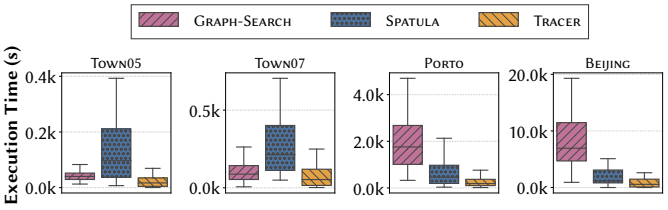


Fig. 10: End-to-end Performance Comparison – The end-to-end execution time of the baselines and TRACER over 4 topologies. Note that the y-axis range is different for each query.

setting, we do not leverage the replay search feature of SPATULA and instead perform an exhaustive search.

- TRACER uses a combination of recurrent network-based camera prediction (§V) and probabilistic adaptive search model (§VI) to process RE-ID queries.
- ORACLE is the ideal system that has access to the ground truth information and hence chooses the best camera and frame to start the search from at each step (§IV).

INCREMENTAL SEARCH OPTIMIZATION. Disabling the incremental search optimization leads to prohibitively longer queries with GRAPH-SEARCH and SPATULA (100× slower). Without this optimization, the systems perform an exhaustive search over all the frames from irrelevant cameras. To better understand the utility of other optimizations in TRACER (*i.e.*, the camera prediction module and the probability update algorithm), we enable the incremental search optimization in all three baselines (GRAPH-SEARCH, SPATULA, and TRACER) across all the experiments.

B. End-to-end Comparison

We first compare the end-to-end execution time of TRACER against the best-performing baselines, GRAPH-SEARCH and SPATULA. As mentioned in §III, we process frames from each camera exhaustively to achieve 100% recall for each baseline. The throughput is averaged over 50 RE-ID queries, with each query executed 20 times to account for variance from the stochastic neighboring camera and start frame selection. The results comparing TRACER to GRAPH-SEARCH and SPATULA are presented in Figure 10.

TRACER VS. GRAPH-SEARCH. TRACER executes RE-ID queries 4.7× faster than GRAPH-SEARCH on average across all 200 queries spanning 4 topologies. TRACER outperforms GRAPH-SEARCH by 7.8× on real-world PORTO and BEIJING topologies. The higher average degree in these topologies (Table II) entails more adjacent cameras to process. So, TRACER benefits from the probability update algorithm that utilizes the high-confidence scores from the camera prediction module to greedily exploit frame windows from the correct cameras. In contrast, GRAPH-SEARCH's random exploration strategy incurs significant overhead before reaching the desired frame in the correct camera. However, for synthetic TOWN05 and TOWN07 topologies with a lower average degree, GRAPH-SEARCH performs reasonably well as its random exploration quickly explores the few neighbors. So, TRACER is only 1.7× faster than GRAPH-SEARCH for these simpler topologies.

TRACER VS. SPATULA. TRACER outperforms SPATULA by 3.9× on average across all 200 queries. This significant gain stems from TRACER's recurrent network effectively capturing complex long-term cross-camera correlations, unlike SPATULA's reliance on less accurate local correlations. Since SPATULA uses static probability scores during all sampling rounds, its weaker prediction model can cause exhaustive yet incorrect exploration of neighboring cameras. Interestingly, TRACER achieves a higher average speedup over SPATULA on synthetic datasets compared to real-world datasets. As seen in Figure 10, SPATULA exhibits high variance on synthetic queries. This variance occurs because SPATULA is sensitive to outliers in synthetic trajectories. Synthetic datasets lack local context, only using a global probability distribution for source and destination cameras (§VII). Thus, SPATULA's local prediction model mispredicts outliers with high-confidence, causing the high variance. This highlights opportunities to improve synthetic data generation methods to better reflect real-world distributions.

SPATULA VS. GRAPH-SEARCH. SPATULA outperforms GRAPH-SEARCH on real-world datasets but is slower on synthetic datasets. As discussed earlier, SPATULA exhaustively processes incorrect neighboring cameras in synthetic datasets due to its inability to handle outliers. In contrast, GRAPH-SEARCH randomly and uniformly explores all the cameras, handling outliers more efficiently. However, GRAPH-SEARCH struggles on real-world datasets with high degree as it must uniformly explore several frame windows in neighboring cameras to arrive at the correct camera and frame.

C. Impact of data distribution

In this experiment, we evaluate the impact of data distribution on the performance of TRACER. Specifically, we compare the performance of TRACER against all the five baselines with changing data skew on the TOWN05 topology. Recall that data skew controls the complexity of long-term cross-camera correlations in synthetic data (§VII). A higher skew level

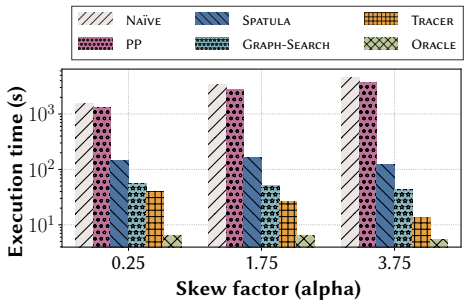


Fig. 11: Varying data skew – The impact of data skew on the end-to-end performance of TRACER and the other baselines in TOWN05.

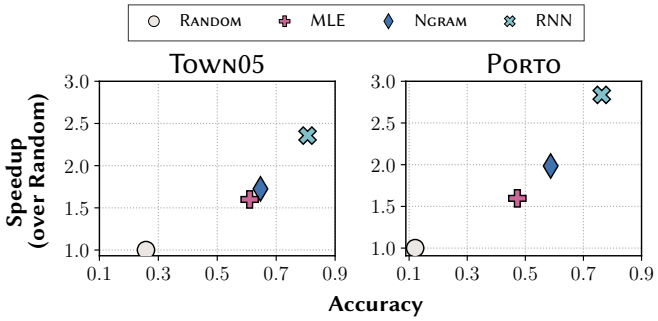


Fig. 12: Camera prediction models – Speedup and accuracy of different camera prediction models in TRACER over synthetic and real-world datasets.

generates vehicles trajectories with strong long-term cross-camera correlations. The results are shown in Figure 11. Note that the y-axis is on a log scale to highlight the differences in the execution time.

NAÏVE AND PP. NAÏVE and PP are the worst-performing methods across all data skews. This is expected since they do not use information about the camera network topology and hence cannot leverage the long-term cross-camera correlations. Additionally, PP only slightly outperforms NAÏVE since the data has a high occupancy rate (Table II).

GRAPH-SEARCH AND SPATULA. GRAPH-SEARCH and SPATULA outperform NAÏVE and PP across all data skews as expected, since they leverage the network topology for camera prediction. GRAPH-SEARCH and SPATULA perform marginally better at higher skew since vehicle trajectories are more predictable. As seen in §VIII-B, GRAPH-SEARCH performs better than SPATULA at all skew levels due to the low average degree of synthetic datasets.

TRACER. TRACER is the best-performing learned method across all data skews. The relative performance of TRACER over the baselines increases with increasing data skew. This can be attributed to the fact that TRACER's probabilistic search model benefits from the high confidence probability outputs of the recurrent network at high data skew. Finally, we note that TRACER's performance is comparable to GRAPH-SEARCH on low skew data. At low skew, the trajectories are not predictable, so the recurrent network in TRACER does not provide a significant advantage over random exploration.

ORACLE. We also show the performance of ORACLE at different data skews. ORACLE can predict the next camera and frame accurately as it has access to the ground truth (§IV). The gap between ORACLE and TRACER reduces with increasing data skew as the learned model in TRACER becomes more accurate. At the highest skew level, the performance of TRACER is only 2.5× lower than ORACLE.

D. Impact of camera prediction model

In this experiment, we evaluate the effectiveness of various models for camera prediction (§V). We report the (1) speedup achieved by the models compared to the random traversal algorithm and (2) the camera prediction accuracy of the models. The results are shown in Figure 12. On aggregate, we observe that the accuracy and speedup of the models vary between real-world and synthetic datasets. The models achieve a higher accuracy on synthetic data since the trajectories are more predictable. However, this accuracy does not translate to a proportionally high speedup since the average degree of the synthetic topologies is low, so incorrect camera predictions are not penalized as much.

N-GRAM MODELS. N-GRAM model achieves lower accuracy compared to RNN. We observe that the N-GRAM model fails to capture the long-term cross-camera correlations in the data. Specifically, we notice that the accuracy of the model drops significantly at larger n (> 3). We identify two reasons for this – (1) the number of possible trajectories grows exponentially with n, increasing the complexity of the learning problem significantly (2) In the training data, long trajectories (larger n) are scarce. In contrast, the RNN model automatically learns the necessary context from a given trajectory, and so, significantly outperforms N-GRAM. Finally, N-GRAM outperforms the frequency estimation method MLE. This demonstrates that considering context from multiple previous cameras improves next-camera prediction accuracy.

E. Size of the camera network

In this experiment, we evaluate the impact of camera network topology size on the accuracy of the camera prediction models in TRACER. Specifically, we compare the performance of TRACER against GRAPH-SEARCH and SPATULA on real-world datasets with varying topology sizes (no.of cameras). We fix the geographical area and increase the number of cameras in the network by sampling more intersections from the map with the same network degree. We report the accuracy of the models at different topology sizes in Figure 13.

We first observe that the random exploration strategy in GRAPH-SEARCH is unaffected by topology size as expected since the decisions are independent of the current trajectory and the network degree is unchanged. The accuracy of TRACER's recurrent network increases with increasing topology size. As the number of cameras in the network increases, more cameras capture the vehicle's movements within the same geographical area, resulting in more fine-grained and accurate vehicle trajectories. This also explains the improving accuracy of SPATULA's neighboring camera frequency estimation model. Importantly, the accuracy gap between TRACER and SPATULA

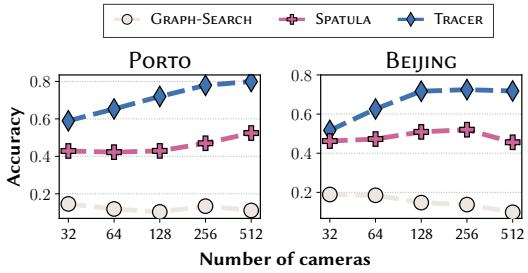


Fig. 13: Changing the camera network size – Impact of camera network size on the accuracy of the learned model in Tracer and the other baselines in Porto and Beijing.

widens for larger topologies as Tracer is better at modeling long-term correlations.

F. Execution speed analysis

Cost breakdown. Figure 14 shows the average cost breakdown per query for different operators in Tracer over 100 queries. We observe that the Re-id operator accounts for the highest cost in Tracer, followed by the object detection models. This is expected since the Re-id model is invoked for every object detected in a given frame. On the other hand, the camera prediction and frame prediction models have a negligible cost (100 ms) since these models only require the current trajectories and not the raw video frames.

Training time. Training time includes the time required to generate the training trajectories and train the models. For synthetic datasets, generating training trajectories only requires 20× the per-query inference cost as discussed in §VII. For real-world deployment, the ground-truth trajectories can be generated offline by running the Re-id pipeline on historical video feeds (§V-D). Our experiments indicate that this offline process only requires ≈10× the per-query inference cost to generate a sufficient number of training trajectories. The RNN model training cost is negligible - training an accurate RNN model takes less than 5 minutes with 25K trajectories from the largest Porto topology.

IX. Related Work

Computer Vision. Multi-camera Re-id is a well-studied problem in the field of Computer Vision [37], [38]. Both traditional [12], [22] and deep learning-based techniques [15], [37], [38] have been proposed for Re-id. These techniques primarily focus on improving the accuracy of the re-identification task, while Tracer focuses on reducing the computational cost.

Video Data Management Systems. Modern VDBMSs [2]–[5], [10] primarily focus on the efficient retrieval of: (1) objects [1]–[3], [6], (2) actions [7]–[9] and (3) object tracks [10], [11]. These systems focus on a single-camera setting and do not consider object relationships across cameras. In contrast, Tracer focuses on efficiently processing Re-id queries - it re-identifies *specific object instances* across a camera network.

Cross-camera analytics. Cross-camera analytics focuses on efficiently processing video queries that span multiple cameras. Prior work in this area has mainly optimized online video analytics in the edge-computing setting [16], [39]–[41]. Chameleon [42] and Spatula [16] leverage localized spatio-temporal cross-camera correlations to reduce profiling cost and latency of live video analytics. In contrast, Tracer focuses on the offline setting and uses a novel probabilistic adaptive query processing framework to reduce Re-id query cost.

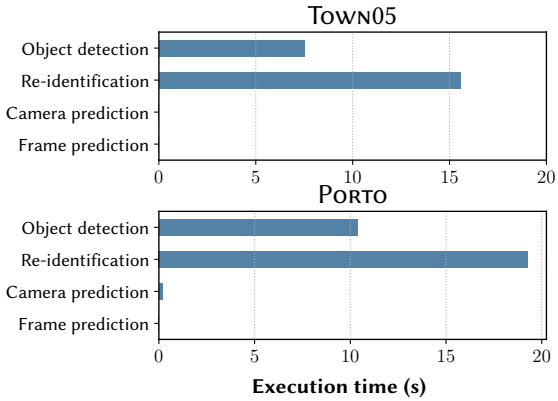


Fig. 14: Cost breakdown – Cost breakdown of different operators involved in processing Re-id queries in Tracer.

Adaptive Query Processing and Sensor Networks. Adaptive query processing has been extensively explored to dynamically reorder query plans and optimize efficiency [43], [44]. It has also been applied to sensor networks to adapt to resource changes and optimize data acquisition [45]–[47]. Tracer draws inspiration from these works to effectively and adaptively select the most cost-efficient cameras at each time step. Additionally, it incorporates a probabilistic adaptive search model, inspired by [48], to optimize Re-id queries.

Spatio-temporal data mining. In spatio-temporal data mining, recurrent neural networks have often been used for spatio-temporal location prediction. These methods apply large recurrent network models to dense location data, such as GPS traces, to capture spatial-temporal context and improve the accuracy of next location prediction [49]–[52]. In contrast, Tracer applies recurrent networks to sparse camera trajectories obtained from video analytics primitives. It focuses on improving Re-id query efficiency and utilizes the recurrent network to optimize the adaptive search process.

X. Conclusion

Efficiently re-identifying and tracking objects across a network of cameras is an important vision problem in practice. In this paper, we presented Tracer, a novel VDBMS tailored for answering such Re-id queries using two key optimizations. First, it trains a recurrent network to model the long-term spatial correlations, thereby improving the camera prediction accuracy. Second, it utilizes a probabilistic adaptive search model that processes camera feeds in incremental windows, and adaptively adjusts the probabilities based on an exploration-exploitation trade-off. Evaluation on three diverse datasets shows that Tracer accelerates Re-id query processing by 3.9× on average compared to Spatula.

REFERENCES

[1] Y. Lu, A. Chowdhery, S. Kandula, and S. Chaudhuri, "Accelerating machine learning inference with probabilistic predicates," in *SIGMOD*, 2018, pp. 1493–1508.

[2] D. Kang, P. Bailis, and M. Zaharia, "Blazeit: optimizing declarative aggregation and limit queries for neural network-based video analytics," *VLDB*, vol. 13, no. 4, pp. 533–546, 2019.

[3] Z. Xu, G. T. Kakkar, J. Arulraj, and U. Ramachandran, "Eva: A symbolic approach to accelerating exploratory video analytics with materialized views," in *SIGMOD*, 2022, pp. 602–616.

[4] M. Daum, E. Zhang, D. He, M. Balazinska, B. Haynes, R. Krishna, A. Craig, and A. Wirsing, "Vocal: Video organization and interactive compositional analytics," in *CIDR*, 2022.

[5] D. Kang, F. Romero, P. Bailis, C. Kozyrakis, and M. Zaharia, "Viva: An end-to-end system for interactive video analytics." CIDR, 2022.

[6] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "Noscope: Optimizing deep cnn-based queries over video streams at scale." *VLDB*, vol. 10, no. 11, pp. 1586–1597, 2017.

[7] P. Chunduri, J. Bang, Y. Lu, and J. Arulraj, "Zeus: Efficiently localizing actions in videos using reinforcement learning," in *SIGMOD*, 2022, pp. 545–558.

[8] E. Zhang, M. Daum, D. He, M. Balazinska, B. Haynes, and R. Krishna, "Equi-vocal: Synthesizing queries for compositional video events from limited user interactions [technical report]," *arXiv preprint arXiv:2301.00929*, 2023.

[9] N. Koudas, R. Li, and I. Xarchakos, "Video monitoring queries," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 10, pp. 5023–5036, 2022.

[10] F. Bastani, S. He, A. Balasingam, K. Gopalakrishnan, M. Alizadeh, H. Balakrishnan, M. Cafarella, T. Kraska, and S. Madden, "Miris: Fast object track queries in video," in *SIGMOD*, 2020, pp. 1907–1921.

[11] F. Bastani and S. Madden, "Otif: Efficient tracker pre-processing over large video datasets," in *SIGMOD*, 2022, pp. 2091–2104.

[12] X. Wang, "Intelligent multi-camera video surveillance: A review," *Pattern recognition letters*, vol. 34, no. 1, pp. 3–19, 2013.

[13] M. Naphade, S. Wang, D. C. Anastasiu, Z. Tang, M. Chang, Y. Yao, L. Zheng, M. S. Rahman, A. Venkatachalapathy, A. Sharma, Q. Feng, V. Ablavsky, S. Sclaroff, P. Chakraborty, A. Li, S. Li, and R. Chellappa, "The 6th ai city challenge," in *CVPRW*. IEEE Computer Society, June 2022, pp. 3346–3355.

[14] Ultralytics, "Yolov5," 2023. [Online]. Available: https://github.com/ultralytics/yolov5

[15] H. Luo, Y. Gu, X. Liao, S. Lai, and W. Jiang, "Bag of tricks and a strong baseline for deep person re-identification," in *CVPRW*, 2019, pp. 0–0.

[16] S. Jain, X. Zhang, Y. Zhou, G. Ananthanarayanan, J. Jiang, Y. Shu, P. Bahl, and J. Gonzalez, "Spatula: Efficient cross-camera video analytics on large camera networks," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2020, pp. 110–124.

[17] "Duke mtmc," https://exposing.ai/duke_mtmc/, accessed: 2023-05-31.

[18] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *CoRL*. PMLR, 2017, pp. 1–16.

[19] B. R. Ardabili, A. D. Pazho, G. A. Noghre, C. Neff, A. Ravindran, and H. Tabkhi, "Understanding ethics, privacy, and regulations in smart video surveillance for public safety," *arXiv preprint arXiv:2212.12936*, 2022.

[20] D. Almeida, K. Shmarko, and E. Lomas, "The ethics of facial recognition technologies, surveillance, and accountability in an age of artificial intelligence: a comparative analysis of us, eu, and uk regulatory frameworks," *AI and Ethics*, vol. 2, no. 3, pp. 377–387, 2022.

[21] D. I. Board, "Ai principles: Recommendations on the ethical use of artificial intelligence by the department of defense," *Supporting document, Defense Innovation Board*, vol. 2, p. 3, 2019.

[22] D. Zapletal and A. Herout, "Vehicle re-identification for automatic video traffic surveillance," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2016, pp. 25–31.

[23] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *CVPR*, vol. 1. Ieee, 2005, pp. 886–893.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[25] M. Risdal, "New york city taxi trip duration," 2017. [Online]. Available: https://kaggle.com/competitions/nyc-taxi-trip-duration

[26] P. F. Brown, V. J. Della Pietra, P. V. Desouza, J. C. Lai, and R. L. Mercer, "Class-based n-gram models of natural language," *Computational linguistics*, vol. 18, no. 4, pp. 467–480, 1992.

[27] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[28] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[30] A. Suprem, J. Arulraj, C. Pu, and J. Ferreira, "Odin: automated drift detection and recovery in video analytics," *VLDB*, vol. 13, no. 12, pp. 2453–2465, 2020.

[31] B. Haynes, A. Mazumdar, M. Balazinska, L. Ceze, and A. Cheung, "Visual road: A video data management benchmark," in *SIGMOD*, 2019, pp. 972–987.

[32] F. Herzog, J. Chen, T. Teepe, J. Gilg, S. Hörmann, and G. Rigoll, "Synthehicle: Multi-vehicle multi-camera tracking in virtual cities," in *WACV*, 2023, pp. 1–11.

[33] Epic Games, "Unreal engine." [Online]. Available: https://www.unrealengine.com

[34] "Carla agents," https://carla.readthedocs.io/en/0.9.14/adv_agents/, accessed: 2023-05-31.

[35] L. Moreira-Matias, J. Gama, M. Ferreira, J. Mendes-Moreira, and L. Damas, "Predicting taxi–passenger demand using streaming data," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1393–1402, 2013.

[36] Y. Zheng, X. Xie, W.-Y. Ma *et al.*, "Geolife: A collaborative social networking service among user, location and trajectory." *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 32–39, 2010.

[37] C. Liu, Y. Zhang, H. Luo, J. Tang, W. Chen, X. Xu, F. Wang, H. Li, and Y.-D. Shen, "City-scale multi-camera vehicle tracking guided by crossroad zones," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 4129–4137.

[38] K. Shim, S. Yoon, K. Ko, and C. Kim, "Multi-target multi-camera vehicle tracking for city-scale traffic management," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 4193–4200.

[39] H. Guo, S. Yao, Z. Yang, Q. Zhou, and K. Nahrstedt, "Crossroi: cross-camera region of interest optimization for efficient real time video analytics at scale," in *Proceedings of the 12th ACM Multimedia Systems Conference*, 2021, pp. 186–199.

[40] H. B. Pasandi and T. Nadeem, "Convince: Collaborative cross-camera video analytics at the edge," in *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2020, pp. 1–5.

[41] K. Chen, Y. Zhu, Z. Han, and X. Wang, "Adaptive cross-camera video analytics at the edge," in *2022 IEEE 19th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*. IEEE, 2022, pp. 394–402.

[42] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *SIGCOMM*, 2018, pp. 253–266.

[43] R. Avnur and J. M. Hellerstein, "Eddies: Continuously adaptive query processing," in *SIGMOD*, 2000, pp. 261–272.

[44] A. Deshpande, Z. Ives, V. Raman *et al.*, "Adaptive query processing," *Foundations and Trends® in Databases*, vol. 1, no. 1, pp. 1–140, 2007.

[45] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: an acquisitional query processing system for sensor networks," *ACM Transactions on database systems (TODS)*, vol. 30, no. 1, pp. 122–173, 2005.

[46] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks," in *VLDB*, 2004, pp. 588–599.

[47] A. Deshpande, C. Guestrin, W. Hong, and S. Madden, "Exploiting correlated attributes in acquisitional query processing," in *ICDE*. IEEE, 2005, pp. 143–154.

[48] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong, "Using probabilistic models for data management in acquisitional environments." in *CIDR*, vol. 5, 2005, pp. 317–328.

[49] Q. Liu, S. Wu, L. Wang, and T. Tan, "Predicting the next location: A recurrent model with spatial and temporal contexts," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[50] A. Al-Molegi, M. Jabreel, and B. Ghaleb, "Stf-rnn: Space time features-based recurrent neural network for predicting people next location," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2016, pp. 1–7.
[51] Y. Xiao and Q. Nian, "Vehicle location prediction based on spatiotemporal feature transformation and hybrid lstm neural network," *Information*, vol. 11, no. 2, p. 84, 2020.
[52] B. Altaf, L. Yu, and X. Zhang, "Spatio-temporal attention based recurrent neural network for next location prediction," in *2018 IEEE International conference on big data (Big Data)*. IEEE, 2018, pp. 937–942.