Learning Road Network Index Structure for Efficient Map Matching

Zhidan Liu, *Member, IEEE*, Yingqian Zhou, Xiaosi Liu, Haodi Zhang, *Member, IEEE*, Yabo Dong, Dongming Lu, and Kaishun Wu, *Fellow, IEEE*

***Abstract*—Map matching aims to align GPS trajectories to their actual travel routes on a road network, which is an essential pre-processing task for most of trajectory-based applications. Many map matching approaches utilize Hidden Markov Model (HMM) as their backbones. Typically, HMM treats GPS samples of a trajectory as observations and nearby road segments as hidden states. During map matching, HMM determines candidate states for each observation with a fixed searching range, and computes the most likely travel route using the *Viterbi* algorithm. Although HMM-based approaches can derive high matching accuracy, they still suffer from high computation overheads. By inspecting the HMM process, we find that the computation bottleneck mainly comes from improper candidate sets, which contain many irrelevant candidates and incur unnecessary computations. In this paper, we present `LiMM` – a learned road network index structure for efficient map matching. `LiMM` improves existing HMM-based approaches from two aspects. First, we propose a novel learned index for road networks, which considers the characteristics of road data. Second, we devise an adaptive searching range mechanism to dynamically adjust the searching range for GPS samples based on their locations. As a result, `LiMM` can provide refined candidate sets for GPS samples and thus accelerate the map matching process. Extensive experiments are conducted with three large real-world GPS trajectory datasets. The results demonstrate that `LiMM` significantly reduces computation overheads by achieving an average speedup of 11.7× than baseline methods, merely with a subtle accuracy loss of 1.8%.**

***Index Terms*—GPS Trajectory, hidden markov model, learned index, map matching, road network.**

Received 22 October 2023; revised 23 June 2024; accepted 19 October 2024. Date of publication 23 October 2024; date of current version 26 November 2024. This work was supported in part by the National Natural Science Foundations of China under Grant 62172284 and Grant U2001207, in part by Guangdong Basic and Applied Basic Research Foundation under Grant 2022A1515010155, in part by the Guangdong Provincial Key Lab of Integrated Communication, Sensing and Computation for Ubiquitous Internet of Things under Grant 2023B1212010007, in part by the Project of DEGP under Grant 2023KCXTD042 and Grant 2021ZDZX1068, and in part by Guangzhou Municipal Science and Technology Project under Grant 2023A03J0011. Recommended for acceptance by M. Zhang. *(Corresponding author: Zhidan Liu.)*

Zhidan Liu is with the INTR Thrust, System Hub, The Hong Kong University of Science and Technology, Guangzhou 510000, China (e-mail: zhidanliu@hkust-gz.edu.cn).

Yingqian Zhou, Xiaosi Liu, and Haodi Zhang are with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China (e-mail: zhouyingqing2020@email.szu.edu.cn; liuxiaosi2022@email.szu.edu.cn; hdzhang@szu.edu.cn).

Yabo Dong and Dongming Lu are with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310058, China (e-mail: dongyb@zju.edu.cn; ldm@zju.edu.cn).

Kaishun Wu is with the DSA Thrust and IoT Thrust, Information Hub, The Hong Kong University of Science and Technology, Guangzhou 510000, China (e-mail: wuks@hkust-gz.edu.cn).

Digital Object Identifier 10.1109/TKDE.2024.3485195

I. Introduction

With the increasing popularization and application of positioning technologies and devices, e.g., GPS sensors, massive GPS trajectories have been collected. Due to the intrinsic inaccuracy of positioning systems, however, such raw GPS trajectories need to be well pre-processed before being used [63]. As one of the most important pre-processing tasks, map matching aims to determine the actual travel route of a given GPS trajectory by aligning the GPS location sequence with an underlying road network. These matched trajectories then can be safely exploited for many trajectory-based applications, such as navigation [60], traffic sensing [34], traffic prediction [35], digital map updating [6], and travel time prediction [57].

Tremendous map matching approaches have been proposed [7], while many of them [8], [19], [23], [37], [41], [49], [54] have been developed based on the Hidden Markov Model (HMM), which is good at modeling the sequence of GPS samples by incorporating additional features, e.g., road connectivity and travel directions. In general, those approaches regard GPS samples as *observations* and nearby road segments as *hidden states* in HMM. Specifically, they determine candidate states for each observation by querying the road network within a searching rage γ, and then compute the most likely travel route by utilizing the *Viterbi* algorithm [41]. Although HMM can derive high map matching accuracy [14], [45], it incurs extremely huge computation overheads. For instance, HMM has to involve about $n \times k^2$ computations of the shortest paths [47], where n is the number of observations in a trajectory and k is the average number of candidate states for these observations.

Valuable efforts have been made to reduce inference time of HMM-based map matching approaches by exploiting parallel computing frameworks [3], [58] or some advanced deep learning models [26], [45], [47]. However, these approaches either require expensive hardware resources, e.g., clustered machines, or heavily depend on a large amount of well-labeled trajectory data, since the deep learning models are usually data-hungry and data-sensitive [14].

By inspecting the HMM process, we find the key to cutting down computation overheads is to reduce the candidate states for each observation. Nevertheless, existing HMM-based approaches primarily rely on traditional index structures, e.g., *R-tree* [18], to index road network data, which may return many irrelevant road segments and thus are inefficient. Additionally, these works adopt a fixed searching range γ to search candidate states for all observations [41], [54]. In fact, GPS positioning errors differ greatly across various areas of a city. Specifically, GPS performs poorly in some urban canyons, while achieving accurate positioning in other urban areas [52]. A fixed searching range is thus less effective in practice.

To address these limitations, we present `LiMM`, a Learned road network index structure for efficient Map Matching. `LiMM` is compatible with existing HMM-based map matching approaches, and improves their computation efficiency while ensuring the matching accuracy by providing refined candidate states. We achieve this attractive target through two ingenious designs. First, we propose a novel learned index structure that is particularly tailored to road network data by considering the characteristics of road segments. Specifically, we have devised a scaling method, which partitions the road network into regular hexagons and then maps road segment data into one-dimensional keys given their geographical and directional information. We sort these keys and then employ a hierarchy of simple machine learning models to approximate their distribution. Based on these learned index models, `LiMM` supports typical range queries for the map matching task.

Second, we further enhance `LiMM`'s query performance for map matching with a reinforcement learning (RL) [27] based adaptive searching range mechanism. Rather than adopting a fixed γ for all samples, we separately train a RL model for each hexagon by leveraging matched trajectory samples. By interacting with the learned road network index for training samples, RL models record the query experiences of various γ values, which are measured with a reward function, and can pick the suitable searching range γ for each incoming query. As a result, range query with adaptive γ can effectively filter out irrelevant candidates. Furthermore, we propose a coverage-oriented trajectory selection method, which heuristically chooses a subset of raw trajectories for map matching, to quickly initialize the RL models for all hexagons.

Based on the above designs, `LiMM` can effectively support various HMM-based map matching approaches. Given a GPS sample, `LiMM` chooses an appropriate searching range γ for the range query, and returns a refined candidate set to conduct efficient and accurate map matching.

We summarize our major contributions as follows.

- We have identified the computation bottleneck of HMM-based map matching approaches, and present `LiMM` to address their limitations with improved computation efficiency and ensured matching accuracy.
- We devise a novel learned index structure particularly for the road network data by considering the characteristics of road segments.
- We propose an adaptive searching range mechanism for HMM-based map matching, which can determine the suitable searching ranges for GPS samples based on their locations to derive refined candidate sets.
- We conduct extensive experiments using three large real-world trajectory datasets. Experimental results demonstrate that `LiMM` significantly outperforms the baseline methods in terms of matching time with an average speedup of $11.7\times$, yet with a subtle accuracy loss of only 1.8%.[1]

The rest of this paper is organized as follows. Section II introduces the preliminary of map matching and discusses the limitations of existing approaches. Section III presents the overview of `LiMM`, followed by detailed designs of learned road network index and adaptive searching range mechanism in Section IV and Section V, respectively. We evaluate `LiMM` in Section VI, and review the related works in Section VII. Finally, Section VIII concludes this paper.

II. Background and Motivation

In this section, we first introduce basic concepts and define the map matching problem. Then, we describe the general process of Hidden Markov Model (HMM) based map matching approaches, and further analyze their limitations.

A. Problem Definition

Definition 1. (GPS Sample): A GPS sample g_i is denoted by a triplet $< t, lat, lng >$, indicating that the object of interest was located at latitude lat and longitude lng at timestamp t.

Definition 2. (GPS Trajectory): A GPS trajectory $\mathcal{T}_j$ is a sequence of time-ordered GPS samples, denoted by $\mathcal{T}_j = \{g_1, g_2, \ldots, g_{|\mathcal{T}_j|}\}$, where $|\mathcal{T}_j|$ indicates the trajectory length.

Definition 3. (Road Network): A road network (also known as road map) is modeled as a directed graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where each vertex $v \in \mathcal{V}$ represents an intersection or a road end, and each edge $e \in \mathcal{E}$ represents a directed road segment.

Definition 4. (Travel Route): A travel route $\mathcal{R}_j$ is a sequence of connected edges on graph $\mathcal{G}$, i.e., $\mathcal{R}_j = \{e_1, e_2, \ldots, e_{|\mathcal{R}_j|}\}$, where $|\mathcal{R}_j|$ is the number of edges on route $\mathcal{R}_j$. Noting that the end point of e_i is the start point of e_{i+1}.

Due to the measurement errors of GPS devices, GPS positioning is not precise [52]. Thus, we have to perform a procedure of map matching to pre-process GPS trajectories for supporting various trajectory-based applications [63].

Definition 5. (Map Matching): Given a road network $\mathcal{G}$ and a GPS trajectory $\mathcal{T}_j$, map matching aims to determine the most likely travel route $\mathcal{R}_j$ that represents the sequence of road segments traveled by trajectory $\mathcal{T}_j$.

Fig. 1 demonstrates a map matching example, where the input GPS trajectory $\mathcal{T}$ [2] is transformed to a travel route.

B. HMM-Based Map Matching

Although there exist many map matching approaches [7], HMM is one of the most widely used map matching models. Because HMM is not only good at modeling the sequence of GPS samples, but also flexible and robust to incorporate auxiliary data, including the road network topology and user's mobility information, e.g., travel speed and direction.

[1] `LiMM` is open-sourced at https://github.com/SZU-BDUC/LiMM. We hope that `LiMM` can benefit the community and serve as an inspiration for future research endeavors.

[2] We will omit the subscript if the context is clear.

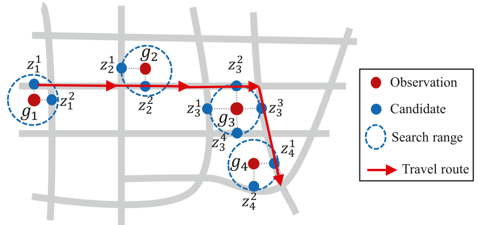


Fig. 1. An example of HMM-based map matching, where a GPS trajectory $\mathcal{T} = \{g_1, g_2, g_3, g_4\}$ is map matched to recover its actual travel route (i.e., the red line) on the road network.

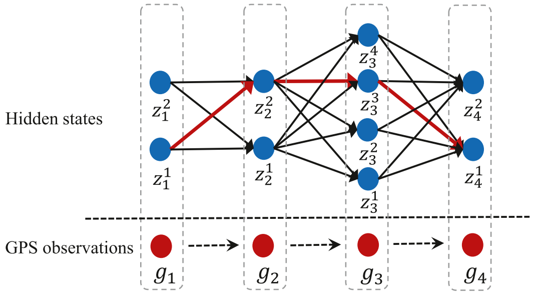


Fig. 2. *Viterbi* inference procedure for trajectory $\mathcal{T}$ shown in Fig. 1.

HMM treats each GPS sample of a trajectory as the *observation*, and the object's actual location on the road, which is to be inferred, as the *hidden state*. Due to GPS measurement errors, all road segments near an observation could potentially be the object's actual located road (state), each of which has a probability (i.e., *emission probability*). Along the timestamps in a trajectory, the transition between two consecutive observations is conducted by the travel probability (i.e., *transition probability*) between their candidate states. The objective is thus to find the optimal path that can properly connect one candidate state in every observation. The object's actual travel route can be inferred by the *Viterbi* algorithm by leveraging the idea of dynamic programming. In general, HMM-based map matching approaches transform a trajectory to its actual travel route through three stages, i.e., *candidate preparation*, *transition calculation*, and *Viterbi inference*. Next, we will follow the typical setting in HMM-based map matching [41] to explain each stage.

(1) Candidate preparation: To tolerate GPS positioning errors, HMM considers all road segments that are within or intersected with a *searching circle*, centered at the observation $g_t \in \mathcal{T}$ with a radius of γ, as the candidates. Specifically, HMM regards the projection z_t^i of g_t to each candidate road segment e_i as a hidden state, and all such hidden states form the candidate set $\mathbb{C}_t$ for g_t. For example, we can get a candidate set $\mathbb{C}_1 = \{z_1^1, z_1^2\}$ for observation g_1 in Fig. 1 by conducting a range query on the road network.

By modeling GPS noises as zero-mean Gaussian distribution, the emission probability of candidate state $z_t^i \in \mathbb{C}_t$ is defined as

$$p(g_t|z_t^i) = \frac{1}{\sqrt{2\pi}\sigma_z} e^{0.5\left(\frac{dist(z_t^i, g_t)}{\sigma z}\right)^2}, \tag{1}$$

where σ_z is the standard deviation of GPS measurement errors, and $dist(z_t^i, g_t)$ returns the distance between observation g_t and candidate state z_t^i. As a result, these candidate states closer to the observation will have larger emission probabilities than those far away.

(2) Transition calculation: For any two consecutive observations, there may exist many combinations for their candidate states. By assuming that the road network distance between two hidden states should be similar to the geographic distance between their observations, the transition probability between state z_{t-1}^i of observation g_{t-1} and state z_t^j of observation g_t is calculated as

$$p(d_t) = \frac{1}{\eta} e^{\frac{-d_t}{\eta}}, \tag{2}$$

$$d_t = \left| dist(g_{t-1}, g_t) - route\left(z_{t-1}^i, z_t^j\right) \right|, \tag{3}$$

where $dist(g_{t-1}, g_t)$ calculates the geographic distance between observation g_{t-1} and g_t, while $route(z_{t-1}^i, z_t^j)$ computes the road network distance between the two candidate states. Parameter η describes the difference between route distances and geographic distances, and can be estimated from real trajectory dataset [41]. In practice, Dijkstra or A^* algorithm can be used to compute the shortest path between two locations [54].

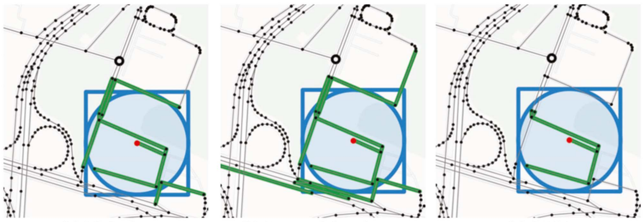
(3) Viterbi inference: Given candidate states for all observations in a trajectory $\mathcal{T}$, along with their associated emission probabilities and transition probabilities, the *Viterbi* algorithm makes use of dynamic programming to infer $\mathcal{T}$'s actual travel route, which visits a sequence of candidate states linked by road segments. The route having the highest accumulated product value of emission probabilities and transition probabilities is finally identified as the optimal travel route for $\mathcal{T}$.

Fig. 2 shows the hidden states for all the four observations in trajectory $\mathcal{T} = \{g_1, g_2, g_3, g_4\}$, and the *Viterbi* inference procedure between these states. The red arrows represent the final inference result, which indicates the actual travel route is the most likely to be the one sequentially linking candidate state z_1^1, z_2^2, z_3^3, and z_4^1.

Many HMM-based map matching approaches have already been proposed to pre-process GPS trajectories [7], [25]. These works usually follow the same definition of emission probability, i.e., (1), but mainly differ in the definition of transition probability by considering different travel preferences, e.g., travel speeds [21], moving directions [8], and route choices [59]. They incorporate such auxiliary information to optimize HMM modeling, yet without changing underlying backbone.

C. Motivation: Road Network Index Matters

Despite having gained a great success on obtaining accurate map matching results, HMM severely suffers from huge computation overheads, which limits its adoptions in some time-sensitive applications [17]. Formally, given a trajectory $\mathcal{T}$ consisting of n GPS samples, the computation complexity of HMM-based map matching could be $\mathcal{O}(n \cdot k^2 \cdot \tau)$, where k is



(a) NoIndex (b) R-tree / Quad-tree (c) LISA

Fig. 3. Visualization of candidate road segments retrieved by different index structures, which search candidates using the blue circle or rectangle.

the average number of candidate states for GPS samples and τ is the computation cost for transition calculation between any two consecutive candidate states. In particular, τ is dominated by the shortest path computation that would be a huge cost over a large-scale road network graph $\mathcal{G}$. Therefore, HMM-based map matching approaches have to conduct $n \cdot k^2$ times of shortest path computation for trajectory $\mathcal{T}$, resulting in a significant response delay.

By analyzing the HMM-based map matching procedure, we find that the number k of candidate states is highly relevant to the computation cost. Existing HMM-based approaches usually build an index structure, e.g., *R-tree* [4] or *Quad-tree* [15], to organize road network graph $\mathcal{G}$, and perform a range search over the pre-built index with a predefined parameter γ to retrieve candidate states for each GPS sample. Therefore, we can identify two factors in determining k, namely *index structure* and *searching range* γ. We conduct experiments to investigate their influences by using a representative HMM-based map matching approach [41]. More details about the experimental settings can be found in Section VI-A.

Impact of index structure: We organize the road network graph $\mathcal{G}$ by using various index structures, including *NoIndex*, *R-tree* [18], *Quad-tree* [15], and *LISA* [30]. Specifically, *NoIndex* represents the range searches over the road network $\mathcal{G}$ without index structure, and *LISA* is an emerging learned index structure that uses machine learning models to generate an optimized data layout and provides efficient search for spatial datasets [30]. In this experiment, we fix the searching range $\gamma = 50\,\text{m}$ for all the index structures.

For each index structure, we perform candidate searching for 2000 GPS trajectories collected in Porto city, and report the statistics. The average numbers of candidate states retrieved by *NoIndex*, *R-tree*, *Quad-tree*, and *LISA* are 16, 20, 20, and 12, respectively. We find that *R-tree* and *Quad-tree* generally obtain more candidates than *NoIndex*, while *LISA* has the fewest candidates. A road segment is usually represented as a pair of start and end vertices, and traditional *R-tree* and *Quad-tree* index road segments using rectangles, e.g., the minimum bounding rectangles (MBRs). However, rectangles are redundant in the spatial space, and they inevitably return irrelevant candidate road segments.

To better understand the performance of these indexes, we visualize the query results of *NoIndex*, *R-tree*, *Quad-tree*, and *LISA* in Fig. 3 for the same GPS sample. We find that these indexes have returned different candidate sets. Specifically, *R-tree* and *Quad-tree* return the same candidates, some of which are irrelevant, due to the spatial redundancy of rectangles. On the contrary, *LISA* only returns road segments that have both endpoints within the searching circle, and thus may miss the "right" road segment. Compared to traditional indexing techniques, learned index, e.g., *LISA* [30], is storage-efficient and speedy in query processing [38]. However, existing learned index methods are not specially designed for the road network data, and cannot get the accurate query results.

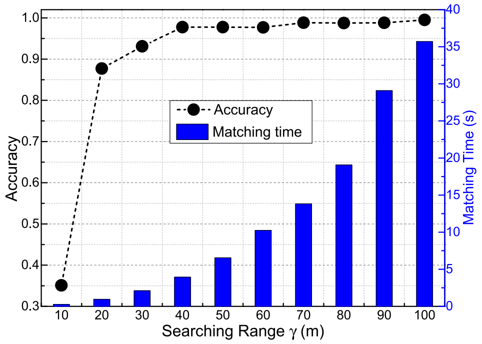


Fig. 4. Impact of searching range γ on map matching performance.

Impact of searching range γ: In addition to the index structures, searching range γ can also affect candidate state query. Taking *R-tree* as the index structure, we conduct experiments using 2000 GPS trajectories from Porto city to examine the impact of γ by varying its values from 10 m to 100 m. Fig. 4 shows the experimental results. When γ is small, the candidate set is small and may miss the "right" road segment, resulting in low matching accuracy. When searching candidate states with a larger γ, we obtain more candidates that will include the correct one with a high probability. The larger γ can help to increase map matching accuracy, while more candidates also incur longer matching time, as shown in Fig. 4.

Existing map matching approaches determine the searching range γ by analyzing the positioning errors of massive trajectories [41], [47], and adopt a fixed γ to retrieve candidate states for all samples. However, GPS positioning performance varies in different areas, leading to distinct GPS errors. In general, GPS could work well in most urban areas, but has poor positioning performance in some places, e.g., urban canyons. For example, by analyzing the trajectory data of GPS-equipped buses, a recent study [36] reports that the average GPS error is 9.1 m in urban areas, while the error could reach 50 m in urban canyons. To ensure the map matching accuracy, existing approaches typically use a conservative searching range, e.g., $\gamma = 50\,\text{m}$, and as a result, they get many irrelevant candidates for samples that originally have small positioning errors.

Takeaways: Existing HMM-based approaches severely suffer from huge computation overheads, and our experiments validate that their computation inefficiency mainly stems from inappropriate candidate states, due to inefficient index structures and the conservative searching range. These irrelevant candidate states will incur substantial unnecessary computations, and thus significantly increase the matching time.

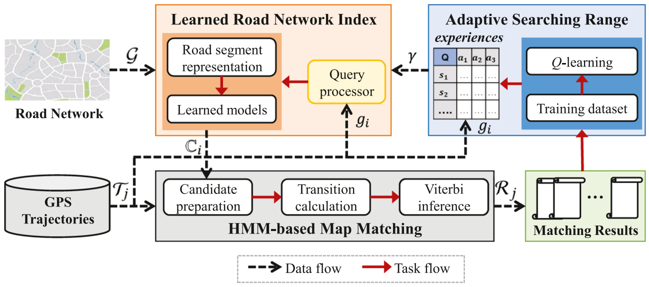


Fig. 5. Framework of LiMM.

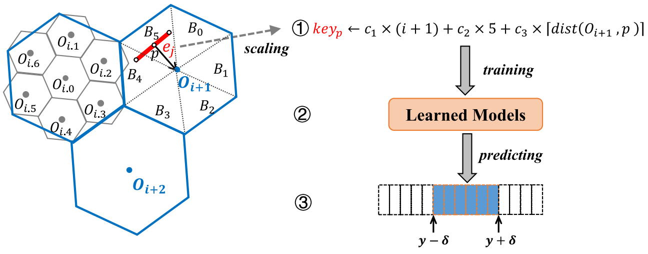


Fig. 6. Illustration of learned index models for the road network.

In response, we present LiMM, a solution with *(i) a novel learned road network index structure*, which is tailored for road network data for fast map matching; and *(ii) an adaptive searching range mechanism* that can adjust γ across different areas based on GPS error distribution. Based on the two key designs, LiMM can provide refined candidate states for existing HMM-based approaches and greatly improve their computation efficiency.

III. Overview of LiMM

Architecture: Fig. 5 illustrates the framework of LiMM, which consists of two key modules, i.e., *learned road network index* and *adaptive search range*. At the high-level, LiMM takes road network graph $\mathcal{G}$ and GPS trajectories as input and enhances existing HMM-based map matching approaches to process each trajectory in an accurate and efficient way.

- The *learned road network index* module (Section IV) aims to index road network data and retrieve candidate states for each GPS sample. This module represents each road segment $e \in \mathcal{E}$ as unique and sortable keys through a novel scaling method by considering both geographical and direction information of e. Then, LiMM builds a set of machine learning models to approximate the order of these scaled keys. In addition, *query processor* of this module can conduct the range query, given a GPS sample g_i and a searching range γ, on the learned models, and return a key set $\mathbb{C}_i$ to HMM-based map matching approaches as refined candidate states.
- The *adaptive searching range* module (Section V) learns appropriate searching ranges for different locations according to their GPS error distribution by analyzing the already matched GPS trajectories. To speedup the learning, LiMM carefully selects a subset of GPS trajectories that can cover as many road segments as possible for the initial map matching, and then exploits the matched trajectory dataset to train a reinforcement learning model via *Q-learning*. The learned searching ranges are stored in Q-tables as the *experiences* for future uses. Once a GPS sample g_i is coming, this module picks up a searching range γ given g_i's location and sends the γ value to the learned road network index module for range query execution. Noting that these experiences can be continuously updated by learning from newly matched trajectories. Therefore, the impact of GPS error variations due to environmental changes, e.g., building demolitions, can be eliminated.

Workflow: Assuming that both learned index models and Q-table experiences have been successfully initialized, LiMM-enhanced map matching works as follows. Given a GPS trajectory $\mathcal{T}_j = \{g_1, g_2, \ldots, g_{|\mathcal{T}_j|}\}$, each GPS sample $g_i \in \mathcal{T}_j$ is sent to the adaptive searching range module to obtain a suitable parameter γ. Using g_i and γ, the query processor retrieves a set $\mathbb{C}_i$ of keys as the query result from the learned models. The HMM-based map matching approach regards $\mathbb{C}_i$ as candidate states for g_i, and derives candidate states for other samples in $\mathcal{T}_j$ following the same operations. After all candidates have been prepared, the map matching approach executes transition calculation and *Viterbi* inference, and recovers the actual travel route $\mathcal{R}_j$. At last, $\mathcal{T}_j$ and $\mathcal{R}_j$ are saved in the database and used to update the experiences via reinforcement learning later.

IV. Learned Road Network Index

As shown in Fig. 5, the learned road network index module of LiMM consists of three components, i.e., *road segment representation*, *learned models*, and *query processor*.

Furthermore, as illustrated in Fig. 6, we present the process by which LiMM represents road segments as keys and employs machine learning models to approximate the distribution of sorted keys. LiMM conducts a range query on the learned index models when given a GPS sample and a searching range. The outcome of this range query is a set of keys representing candidate road segments for the given sample in map matching tasks.

A. Road Segment Representation

One important step of learned index is to transform multi-dimensional data, where road segments can be represented as four-dimensional data with the coordinates of start and end vertices, into one-dimensional keys, which can be easily sorted and approximated. To this end, a scaling method is required to group these multi-dimensional data points according to their distribution and project them into one-dimensional keys. The derived keys should preserve the proximity of road segments, i.e., spatially adjacent road segments have similar keys, and more importantly each key corresponds to one and the only one road segment.

Inspired by the idea of *iDistance index* [24], we propose a novel scaling method that can assign one-dimensional keys to road segments by leveraging their geographical and directional information. Specifically, *iDistance index* is a well-known reference-based scaling method, which maps one multi-dimensional data point p into a one-dimensional key based on $key = c \times i + dist(o_i, p)$, where o_i is the closest reference point to p and i is the reference index, c is a constant parameter to partition the multi-dimensional data points into predefined ranges, and $dist(o_i, p)$ calculates the distance between reference o_i and p. In LiMM, we extend the idea of *iDistance index* by considering the characteristics of road segments and mapping them to their keys through the following two operations.

(1) Determining references: To determine a set of suitable reference points for the road network $\mathcal{G}$, we partition $\mathcal{G}$ using regular hexagons with a parameterized side length L. Compared to circles or grids, regular hexagon can be easily operated, and meanwhile it is the closest geometry to a circle. In addition, the distance between the centroid of a hexagon and any data point falling within the hexagon can be well bounded by parameter L. The centroid of each hexagon thus serves as the reference for neighboring data points. To distinguish these reference points, we regard the central hexagon of graph $\mathcal{G}$ as the origin, and assign hexagon IDs in a spiral pattern. The i-th hexagon is denoted by H_i, and its centroid is treated as reference point o_i.

As road segments are not evenly distributed in a city, some hexagons may contain significantly more road segments than others. To guarantee that we can obtain distinguishable keys for all road segments, if hexagon H_i covers too many road segments, e.g., $\geq \lambda$, it will be further divided into seven sub-hexagons with side length as $\frac{L}{\sqrt{7}}$. The sub-hexagons are assigned IDs based on index i. Specifically, the central sub-hexagon is labeled as $i.0$, while other sub-hexagons are spirally assigned IDs from $i.1$ to $i.6$. The left part of Fig. 6 shows how we divide a hexagon and re-number these sub-hexagons. Moreover, each hexagon is divided into six blocks, each of which corresponds to different directions with respect to the hexagon's centroid. The blocks are numbered spirally from B_0 to B_5, as shown in Fig. 6. By doing this, a road segment can be located more precisely by both hexagon and block.

(2) Assigning keys: Using the hierarchical hexagons and the reference points, any road segment $e \in \mathcal{E}$ can be coarsely described by three parts: hexagon H_i, direction block B_j of H_i, and distance to centroid o_i of H_i. However, a road segment may pass through more than one block of a hexagon or even several hexagons, leading to multiple keys for one road segment. To compute all keys for road segment e, we complete the key assignments as following steps:

① In addition to the start and end vertices of road segment e, we generate b virtual points by sampling along e, such that these virtual points can equally divide segment e. In practice, it is extremely rare for two road segments to completely overlap. Therefore, aside from the start and end vertices, any two road segments are unlikely to share common virtual points. To mitigate the issue of common points like the start or end vertex, we associate the start vertex only with its corresponding road segment e. Consequently, the $b + 1$ points form a point set $\mathbb{P}_e = \{p_u | u = 1, 2, \ldots, b + 1\}$ for road segment e.

② For each point $p_u \in \mathbb{P}_e$, we find the closest reference point o_i to p_u, and thus determine its locating hexagon H_i. Noting that H_i could be a sub-hexagon. Regarding true north as the zero-degree direction, we calculate a direction θ_u from p_u to o_i, and determine the block index as $B_j = \lceil \frac{\theta_u}{\omega} \rceil$, where ω is a predefined parameter. For each point p_u, we thus obtain a triplet $< p_u, H_i, B_j >$.

③ We classify all triplets into groups, where each group shares the same H_i and B_j. For each group, we find the minimum distance, denoted by $dist(o_i, p)$, between each point p of this triplet group and reference o_i. We can then compute a key for this triplet group as

$$key_e = c_1 \times i + c_2 \times B_j + c_3 \times \lceil dist\left(o_i, p\right) \rceil , \tag{4}$$

where c_1, c_2, and c_3 are parameters that serve to partition data points into predefined ranges, stretching the ranges differently based on their values. By calculating a key for each triplet group, we thus derive a set of keys for e. Different from previous learned index techniques [38], we generate multiple keys for each road segment. This is because each road segment consists of numerous points, and we do not want to miss any candidate road segments when querying the index for each GPS sample.

The appropriate settings for parameters c_1, c_2, and c_3 are contingent upon the distribution of road segments. These parameters are crucial for generating keys without false positives, which are caused by mapping road segments far from each other to the same key. In principle, by properly setting the scaling parameter c_1 and c_2, we can ensure that road segments falling into different hexagons and blocks have keys in distinct ranges. Similarly, with appropriate setting of parameter c_3, we can distinguish the keys of neighboring points belonging to different road segments. Since we have associated the start vertex with its corresponding road segment, any two road segments rarely have common points, ensuring that the key generated by (4) uniquely corresponds to one and only one road segment.

B. Learned Models

Once the scaling keys for all road segments are obtained, we sort these keys and then build learned index models to approximate the distribution of these sorted keys.

Structure: Several learned index structures are available in the literature [38], while we utilize the popular *recursive model index* (RMI) structure [28] to index road segment keys, due to its simplicity and efficiency. Rather than building one single model, RMI builds a hierarchy of models, which searches the position of a given key in multiple stages. Given an input key, the model at each stage provides a prediction for picking a model of the next stage, or the position of the key when the final stage is reached. The multi-stage models approximate the *cumulative distribution function* (CDF) of a sorted array, modeled as $y = \mathcal{F}(x) \times N$, where x is the lookup key, N is the number of keys, $\mathcal{F}(\cdot)$ is the CDF approximator that estimates the proportion of keys less than the particular lookup key x, and y is the estimated position. While imperfect, RMI can guarantee that the lookup key is within the proximity of predicted position y, i.e., $[y - \delta, y + \delta]$ with bounded error δ. An advantage of learned index is that it will never produce an incorrect search bound [38].

We realize the learned road network index using RMI, as illustrated in Fig. 7, which is a three-stage model in accordance

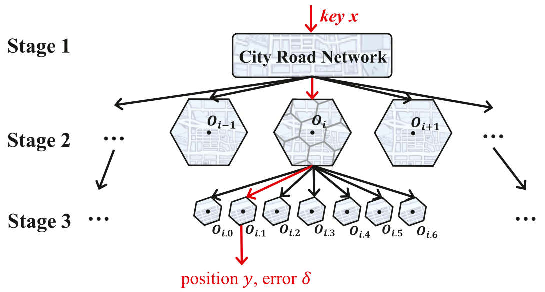


Fig. 7. A hierarchy of models for indexing road segment keys.

with road network partitions. The CDF approximator for road segment keys and their positions is composed of a single stage-one model f_1, a number of stage-two and stage-three models. Intuitively, we exploit the stage-one model to predict which hexagon the input key belongs to, and utilize each stage-two model f_2^i to approximate the CDF of keys within a hexagon. If a hexagon has been divided into sub-hexagons, the RMI model will further make the corresponding stage-two model to predict which sub-hexagon the input key belongs to, and key/position pairs of each sub-hexagon will be modeled by a stage-three model f_3^j. Therefore, the top stage model provides a rough position of the lookup key, while the bottom stage model improves the prediction. As a position y is predicted with a certain error δ that could be known when training the model, we search for the correct position within error bound $[y-\delta, y+\delta]$. Since keys are sorted in an ascending order, binary search can be adopted to speed up the searching.

Training: We implement the model in each stage as a linear regression model. Despite its simplicity, the linear regression models are sufficient to approximate a subset of the given key/position dataset, leading to accurate and fast searching. Once we obtain the keys for road segments, we sort and store them in the memory as <key, position> pairs, which are used as a dataset for index model training. Formally, let $(x, y) \in \mathcal{D}$ be the set of key/position pairs in the dataset, we train the RMI model by adjusting the parameters in each stage model, i.e., f_1, f_2^i, and f_3^j, to minimize the loss:

$$\sum_{(x,y)\in\mathcal{D}} \left(\mathcal{F}(x) - y\right)^2. \tag{5}$$

We train the RMI model in a top-down manner. Specifically, we first train the stage-one model, and fine-tune the predictions by training stage-two models, followed by training over stage-three models if necessary. We thus iteratively train each stage model to build the complete RMI model.

Extensibility: While the road network in a developed city is relatively static with rare constructions or removals of road segments, the RMI model is still extensible in inserting or deleting key/position pairs due to possible road changes. As suggested by [12], we should periodically create additional gaps in the key array to keep a sufficient number of gaps for supporting fast insertions. In general, the insertion or deleting operations require $\mathcal{O}(N+\kappa)$ time, where N is the size of dataset $\mathcal{D}$ and κ is the number of road segment keys to be inserted or deleted.

C. Query Processor

For the map matching task, we mainly concern *range query* on the learned index. Formally, given a GPS sample g and a searching range γ, the query processor is expected to return a set of road segment keys, where the geographical distance between each found road segment and g is no more than γ. `LiMM` processes such a range query in three steps:

① We identify the hexagons (or sub-hexagons) set $\mathbb{H}_g$ that overlaps with the searching circle centered at g with a radius of γ. To this end, we compute the distance between each reference point o_i and g, denoted by d_{go}, and add the associated hexagon (or sub-hexagon) into $\mathbb{H}_g$ if $d_{go} < d_{edge} + \gamma$, where $d_{edge} = \frac{\sqrt{3}}{2}L$ indicates the distance from centroid o_i to any side of hexagon H_i.

② For the overlapping area associated with hexagon $H_i \in \mathbb{H}_g$, we compute the shortest distance $d_{\min}$ and the longest distance $d_{\max}$ from reference point o_i of H_i to the area. Specifically, if $d_{go} \leq \gamma$, the shortest distance $d_{\min} = 0$ because the circle's center g is within the hexagon's range; Otherwise, $d_{\min} = d_{go} - \gamma$. Therefore, we have the following formula for the shortest distance calculation:

$$d_{\min} = \begin{cases} 0, & \text{if } 0 \leq d_{go} \leq \gamma; \\ d_{go} - \gamma, & \text{if } \gamma < d_{go} < d_{edge} + \gamma. \end{cases} \tag{6}$$

Similarly, we have the following formula for the calculation of longest distance $d_{\max}$ through analysis of geometrical relation between hexagon H_i and the searching circle:

$$d_{\max} = \begin{cases} d_{go} + \gamma, & \text{if } d_{go} + \gamma \leq d_{edge}; \\ d_{edge}, & \text{if } d_{go} + \gamma > d_{edge} \text{ and } d_{go} \leq d_{edge}; \\ d_{go} + \gamma, & \text{if } d_{go} > d_{edge} \text{ and } d_{go} + \gamma \leq L; \\ L, & \text{if } d_{go} > d_{edge} \text{ and } d_{go} + \gamma > L. \end{cases} \tag{7}$$

Additionally, by leveraging the reference point o_i and the uppermost and lowermost points of the overlapping area, we can derive the minimum block index $B_{\min}$ and maximum block index $B_{\max}$. According to the key assignment scheme in (4), we estimate the low bound key key_l and upper bound key key_u for this overlapping area using (8) and (9), respectively.

$$key_l = c_1 \times i + c_2 \times B_{\min} + c_3 \times d_{\min} \tag{8}$$

$$key_u = c_1 \times i + c_2 \times B_{\max} + c_3 \times d_{\max} \tag{9}$$

We then separately input key_l and key_u into the learned index models, and obtain the position searching bound $[y_l - \delta_l, y_l + \delta_l]$ and $[y_u - \delta_u, y_u + \delta_u]$ for key_l and key_u, respectively, where y_l and y_u are the predicted positions of key_l and key_u, respectively, and δ_l and δ_u are the corresponding error bounds. At last, we scan the two searching bounds to retrieve the keys of candidate road segments in this overlapping area.

③ We repeat step ② for all hexagons in $\mathbb{H}_g$, and merge these derived key sets to form the final query result $\mathbb{K}_g$, which includes all candidate road segments for sample g.

As one road segment may have multiple keys, we map the keys in $\mathbb{K}_g$ back to their corresponding road segments, which together form the candidate state set $\mathbb{C}_g$ of sample g for the map matching operations, as shown in Fig. 5.

V. Adaptive Searching Range

Rather than adopting a fixed searching range γ for candidate preparation, `LiMM` uses an adaptive γ, which can vary across different areas according to the GPS positioning errors.

A. Searching Range Determination Modeling

A straightforward way to finding suitable γ is to treat the average GPS error of trajectory samples collected within each hexagon H_i as its searching range. However, this approach omits the road network index, and thus may produce inappropriate query results, e.g., missing the "right" candidate or returning too many irrelevant candidates.

To determine the suitable searching range γ for hexagon H_i, we have to extensively query the road network index with different γ values for a given GPS sample and examine the candidate set. The best γ value for each hexagon H_i should find the "right" candidate road segment while minimizing the candidate sets for most GPS samples located within H_i. To this end, we model the searching range determination for hexagon H_i as a Markov Decision Process (MDP) problem, where road network index is viewed as the environment and a given map matching algorithm is the agent. Specifically, our MDP is formally defined as follows:

- *State* s: We treat each possible searching range value as a state. Considering the typical GPS errors in an urban city, we set the maximum searching range value as $\gamma_{\max} = 100\,\text{m}$ [52], and discrete the possible values with a gap of $\Delta = 5$ meters. Therefore, we have in total 20 states, i.e., $\mathcal{S} = [5, 10, 15, \ldots, 95, 100]$.
- *Action* a: Given current state s, the agent may have at most three possible actions to adjust the searching range. Specifically, $a = 1$ indicates to increase s by Δ, $a = -1$ means to decrease s by Δ, while $a = 0$ will keep s unaltered. Noting that, state $s = 5$ has two actions only, i.e., $a = 1$ or 0, while state $s = 100$ also has two actions as $a = -1$ or 0.
- *Reward* r: As the feedback from the environment, a reward r is calculated to evaluate the action given current state. In our problem setup, for a given GPS sample g we prefer the state s that can retrieve a refined candidate state set $\mathbb{C}_g$ over the road network index. In particular, the refined set $\mathbb{C}_g$ must contain the "right" candidate state for g. Therefore, we define the reward function as follows:

$$r = \frac{\phi(\mathbb{C}_g^{s'}, g)}{|\mathbb{C}_g^{s'}|}, \tag{10}$$

where s' is the next state after applying an action on state s, $|\mathbb{C}_g^{s'}|$ is the size of candidate set for GPS sample g given the searching range s'. In addition, $\phi(\mathbb{C}_g^{s'}, g)$ indicates whether set $\mathbb{C}_g^{s'}$ contains the "right" candidate state for g. If it does, $\phi(\mathbb{C}_g^{s'}, g) = 1$; otherwise $\phi(\mathbb{C}_g^{s'}, g) = 0$.

Offline training: The goal of the MDP is to find the optimal strategy, which guides the decision-making at each state based on estimating the state-action value function (i.e., *Q*-function) of the agent. Based on the above MDP modeling, we exploit the reinforcement learning (RL) algorithm [27] to learn the suitable searching ranges for all hexagons. To that end, `LiMM` will employ tabular *Q*-learning to approximate the *Q*-function. *Q*-learning is an off-policy, value-based RL algorithm to learn the value of an action in a particular state, and can produce a state-action value table (i.e., *Q*-table) to indicate the best action for a particular state. Given the input training dataset, *Q*-learning dynamically updates the *Q*-table through exploitation and exploration with the ε-greedy policy [56].

Algorithm 1: Q **-Learning for Hexagon** H_i.

```
Input  : Samples $\mathbb{G}_{H_i}$, learning rate $\alpha$, discount factor
         $\beta$, gap $\Delta$
Output: Learned experiences $Q(s, a)$
1  Initialize $Q(s, a)$ with zeros for all states and actions;
2  for epoch $= 1$ to max-episodes do
3      Initialize state $s = s_{max}$;
4      for step $t = 1$ to $|\mathbb{G}_{H_i}|$ do
5          Retrieve a GPS sample $g \leftarrow \mathbb{G}_{H_i}(t)$;
6          Select an action $a$ with $\varepsilon$-greedy policy;
7          Update state $s' \leftarrow s + \Delta \times a$;
8          Obtain $\mathbb{C}_g^{s'}$ by querying the road network index
             with $g$'s location and searching range $s'$;
9          Compute reward $r$ using Eq. (10);
10         Update $Q(s, a)$ value using Eq. (11);
```

To speedup the model training, we directly take some well map-matched GPS trajectories as the input. More specifically, we make use of GPS samples and their matched road segments to calculate the rewards, as expressed in (10). By classifying those GPS samples into different hexagons based on samples' locations, we then exploit GPS samples of each hexagon H_i, denoted by $\mathbb{G}_{H_i}$, to separately learn a *Q*-table for hexagon H_i.

Algorithm 1 presents the pseudocode of *Q*-learning framework of adaptive searching range determination for hexagon H_i. The algorithm takes samples $\mathbb{G}_{H_i}$ as the input, and outputs the learned *Q*-table for hexagon H_i. At first, the values in *Q*-table are initialized as *zeros*. Then, it continuously updates the *Q*-table for a predefined episodes based on the training data. In each episode, the algorithm initializes state s as the maximum searching range, i.e., $s_{\max}$ (line 3), and then uses all samples to dynamically update the state-action value function (line 4-10). At each step, it retrieves a GPS sample g, and chooses an action a with ε-greedy policy. By interacting with the road network index, the algorithms computes immediate reward using (10). The total *Q*-value is updated as:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \beta \max_{a'} Q(s', a') - Q(s, a)], \tag{11}$$

where α is the learning rate and β is the discount factor.

Online use: Once the *Q*-table for hexagon H_i has been learned, we can get the appropriate searching range γ for GPS samples that fall within H_i to search their candidate states over the road network index. Specifically, according to the historical experiences stored in the *Q*-table, we take γ as the state owning

Algorithm 2: Heuristic Trajectory Selection Method.

```
Input  : Candidate trajectories $\mathbb{T}$, all hexagons $\mathbb{H}$,
         budget $\mathcal{B}$
Output: Selected trajectories $\mathbb{S}$
1  $\mathbb{S} = \emptyset$;
2  while $|\mathbb{S}| < \mathcal{B}$ do
3      $\mathbb{H}_{temp} = \mathbb{H}$;
4      while $\mathbb{H}_{temp}!=\emptyset$ do
5          $H_i = \arg\min_{H_*.lt}\{|H_k.lt|,\ \forall H_k \in \mathbb{H}_{temp}\}$;
6          $\mathcal{T}_j = \arg\max_{\mathcal{T}_*.lh}\{|\mathcal{T}_z.lh|,\ \forall \mathcal{T}_z \in H_i.lt\}$;
7          for $H_k \in \mathcal{T}_j.lh$ do
8              Remove $H_k$ from $\mathbb{H}_{temp}$;
9              Remove $\mathcal{T}_j$ from $H_k.lt$;
10         Add trajectory $\mathcal{T}_j$ into $\mathbb{S}$;
11         Remove trajectory $\mathcal{T}_j$ from $\mathbb{T}$;
12         if $|\mathbb{S}|==\mathcal{B}$ then
13             break;
```

TABLE I
STATISTICS OF THE THREE GPS TRAJECTORY DATASETS

Dataset	Road network ($|\mathcal{V}|, |\mathcal{E}|$)	# of traj.	# of samples
Chengdu	(8319, 16791)	224288	19662685
Porto	(78099, 149781)	1796067	50685451
Shenzhen	(143501, 262425)	914524	27212627

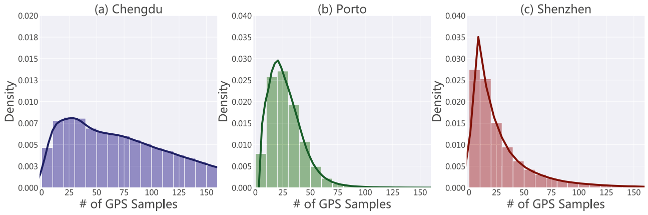


Fig. 8. Density distribution of trajectory lengths for the three datasets.

the highest Q-value, i.e.,

$$\gamma = \arg\max_{s^* \in \mathcal{S}} Q(s^*, a). \tag{12}$$

In principle, such a searching range γ can return refined candidate sets for most GPS samples. While for few special cases that miss the "right" candidate road segment, we can gradually increase γ by Δ for the next candidate searching.

B. Coverage-Oriented Training Trajectory Selection

Since the RL model needs map matched GPS samples for computing Q-tables for all hexagons, it is necessary to carefully select a subset of trajectories for the initial map matching. These matched trajectories then can serve as the bootstrap to quickly initialize the Q-tables, which can speedup the consequent map matching tasks. Moreover, the follow-up map matched trajectory data would be fed into the RL model to further update the experiences in Q-tables.

There are two requirements for the selection of raw trajectories. First, we would like to leverage as few trajectory data as possible to learn the experiences, due to the expensive computation overheads of map matching. Second, GPS samples of those selected trajectories should fully cover all hexagons and be sufficient for each hexagon to learn its Q-table. To this end, we propose a coverage-oriented trajectory selection method, which selects training trajectories in a heuristic manner.

Given the budget $\mathcal{B}$ of raw GPS trajectories to select, our method iteratively selects trajectories that can maximize the hexagon coverage. For each trajectory $\mathcal{T}_j$ in candidate set $\mathbb{T}$, we transform its GPS samples into a sequence of hexagons, which is denoted by $\mathcal{T}_j.lh$, according to the samples' locations. In addition, for each hexagon H_i, we also maintain a list to record the trajectories that pass through hexagon H_i, which is denoted by $H_i.lt$. Based on these information, we run Algorithm 2 to select $\mathcal{B}$ trajectories from $\mathbb{T}$. We iteratively select GPS trajectories to cover all hexagons $\mathbb{H}$, and repeat for multiple times to guarantee that each hexagon can be covered by sufficient trajectory samples. Specifically, in each iteration (line 4-13), we first find the hexagon H_i that owns the fewest pass-by trajectories (line 5), and then select the trajectory $\mathcal{T}_j$ from $H_i.lt$ that has passed the most hexagons (line 6).

VI. PERFORMANCE EVALUATION

A. Experimental Setup

Datasets: We conduct extensive experiments to evaluate LiMM using three real-world GPS trajectory datasets: (*i*) *Chengdu* dataset includes the GPS trajectories of ride-hailing vehicles, which record their status every 1 s, in Chengdu city, China. This dataset was publicly released by Didi's GAIA initiative. (*ii*) *Porto* dataset contains taxi trajectories in Porto city, Portugal. Each taxi reports a GPS sample every 15 seconds. This dataset can be publicly accessed in Kaggle [2]. (*iii*) *Shenzhen* dataset contains GPS trajectories that are generated by buses and taxis with an average sampling rate of 10 seconds in Shenzhen city, China. The dataset is provided by our collaborator for research purpose only. In addition, we have downloaded the road network data from OpenStreetMap (OSM) [1] for the three cities, respectively.

Table I presents statistics about the datasets. Specifically, we have the largest road network in *Shenzhen* dataset, while we have the most trajectories/samples in *Porto* dataset.

We analyze the distribution of trajectory length, i.e., the number of GPS samples in each trajectory, and visualize the results in Fig. 8. We find that *Porto* and *Shenzhen* datasets contain more short trajectories with less than 50 GPS samples, while the density distribution of trajectories in the *Chengdu* dataset is relatively more uniform. We classify GPS trajectories into three categories as *short*, *medium*, and *long* according to their lengths. For performance evaluation, we will randomly select 5000 trajectories from each dataset for the experiments, which consist of *short*, *medium*, and *long* trajectories with the compositions following the density distribution profiled in Fig. 8. Table II shows detailed information about the selected experiment trajectories.

Baselines: We realize a famous HMM-based map matching algorithm [41] as the underlying backbone, which invokes a given indexing method to search candidate states for GPS

TABLE II
COMPOSITION OF SELECTED GPS TRAJECTORIES FOR THE EXPERIMENTS, WHERE *#S* REPRESENTS THE NUMBER OF GPS SAMPLES AND *#T* REPRESENTS THE NUMBER OF TRAJECTORIES

Category	*Chengdu*		*Porto*		*Shenzhen*	
	#S	#T	#S	#T	#S	#T
Short	0∼50	2000	0∼25	2500	0∼50	3500
Medium	51∼100	2000	26∼50	2000	51∼100	1000
Long	>100	1000	>50	500	>100	500
Total	357532	**5000**	143413	**5000**	206576	**5000**

TABLE III
MAJOR PARAMETER SETTINGS, WHERE THE DEFAULT VALUE IS MARKED IN BOLD

Parameter	**Values**
Searching range γ	10, 25, **50**, 75, 100 (m)
Hexagon side length L	250, **500**, 750, 1000 (m)
Block degree threshold ω	15, 30, 45, **60**, 75, 90 $(^\circ)$
Capacity threshold λ	64, 128, **256**, 512, 1024

samples. We compare the performance of `LiMM` with the following indexing methods.

- *Quad-tree* is a spatial index structure [15] that recursively subdivides the two-dimensional space, e.g., a road network $\mathcal{G}$, into four quadrants until the number of spatial objects, e.g., road segments, in each quadrant is smaller than a predefined threshold. To identify candidates for a GPS sample with a searching range γ, *Quad-tree* obtains the quadrants that intersect with the searching circle, and then examines each candidate in the intersected quadrants.
- *R-tree* is another traditional tree-based spatial index structure [18] that groups nearby objects, e.g., road segments, and represents them with their MBRs in the upper level of the tree. Each leaf node of *R-tree* contains only one object. To query candidate road segments for a GPS sample g with searching range γ, *R-tree* first constructs a query rectangle that centers at g with side length as $2 \times \gamma$. Then, *R-tree* traverses the tree and returns these objects, whose MBRs have intersected with the query rectangle.
- *LISA* is the state-of-the-art learned index structure for spatial data [30]. Given an arbitrary spatial dataset, *LISA* maps spatial data into one-dimensional values, and then learns a set of machine learning models to generate the searchable data layout in disk pages. For the given road network data, we transform each road segment into a four-dimensional spatial point with the coordinates of start and end vertices, and then employ *LISA* to build the learned index. *LISA* supports range query by constructing a query rectangle using a GPS sample g and searching range γ as the input. However, spatial points derived by *LISA* need to be mapped back to road segments according to their start/end coordinates.
- `LiMM`*-50* is one variant of our `LiMM`. The only difference with `LiMM` is that this variant adopts a fixed searching range $\gamma = 50\,\text{m}$ to search candidate states.

Performance metrics: To generate ground truth travel routes for the experiment GPS trajectories, we utilize HMM-based map matching algorithm [41], which adopts the *R-tree* index structure with a conservative searching range $\gamma = 100\,\text{m}$, to map raw trajectories to the road network for obtaining their actual travel routes. Following the previous work [45], we define the matching *accuracy* for a given trajectory $\mathcal{T}_i$ as follows:

$$accuracy = \frac{len(\mathcal{R}_i^* \bigcap \bar{\mathcal{R}}_i)}{len(\mathcal{R}_i^*)}, \qquad (13)$$

where $\bar{\mathcal{R}}_i$ and $\mathcal{R}_i^*$ are the mapped travel route by a given method and ground truth travel route for trajectory $\mathcal{T}_i$, respectively. Additionally, $\mathcal{R}_i^* \bigcap \bar{\mathcal{R}}_i$ indicates the correctly matched road segments, and function $len(\cdot)$ calculates the road network distance for a given travel route. This metric favors indexing methods that can return query results containing the correct road segments. Higher values of accuracy indicate that the indexing method more precisely identifies the correct road segments. We also record the end-to-end *matching time* for each trajectory $\mathcal{T}_i$.

To evaluate query processing performance of indexing methods, we utilize the metrics of the *number of candidates*, *hit ratio*, and *query time*. For a given GPS sample g and searching range γ, each indexing method will query the index structure and return a candidate set $\mathbb{C}_g$. Hence, the *number of candidates* is $|\mathbb{C}_g|$, and *query time* is the execution time for an indexing method to obtain $\mathbb{C}_g$ upon receiving the query. The *hit ratio* is defined as the fraction of samples for which a method has retrieved the "right" road segments.

For performance evaluations, we only report the average experimental results for all performance metrics.

Implementation: We implement `LiMM` and other baseline methods in Python 3.7. We have carefully tuned the parameter settings of baselines to achieve their best performance, and fix their searching range $\gamma = 50\,\text{m}$. To evaluate `LiMM`, we examine different sizes of hexagons by varying side length L. If the number of road segments covered by a hexagon exceeds the capacity threshold λ, this hexagon will be divided into sub-hexagons with side length as $\frac{L}{\sqrt{7}}$. Each hexagon (or sub-hexagon) is also divided into blocks every ω degrees. To produce unique keys for road segments, we set the scaling factors in (4) as $c_1 = 10000$, $c_2 = 1000$, and $c_3 = 1$. To train the RL models, we set learning rate $\alpha = 0.1$, discount factor $\beta = 0.8$, and select actions with $\varepsilon = 0.9$ probability. We set the budget $\mathcal{B} = 2000$ for training trajectory selection. Table III shows the major parameter settings.

All experiments are conducted on a server with CPU of Intel(R) Core(TM) i7-10700K 3.80 GHz and memory of 32 GB.

B. Performance Comparison

Accuracy: Table IV summarizes the *matching accuracy* results of all methods over different trajectory categories across the three datasets. We have the following key observations.

(1) In general, each method can achieve a higher matching accuracy when the processed trajectories become longer. This is because the HMM model can infer hidden states more accurately with more observations, i.e., GPS samples.

TABLE IV
OVERALL PERFORMANCE COMPARISON FOR DIFFERENT INDEXING METHODS ON THE *MATCHING ACCURACY*

Method	*Chengdu*				*Porto*				*Shenzhen*			
	Short	Medium	Long	***Overall***	Short	Medium	Long	***Overall***	Short	Medium	Long	***Overall***
Quad-tree	0.979	0.988	0.995	***0.989***	0.968	0.987	0.992	***0.983***	0.984	0.994	0.997	***0.992***
R-tree	0.979	0.988	0.995	***0.989***	0.968	0.987	0.992	***0.983***	0.984	0.994	0.997	***0.992***
LISA	0.733	0.761	0.786	***0.768***	0.709	0.715	0.718	***0.715***	0.496	0.533	0.545	***0.527***
LiMM-***50***	0.976	0.986	0.992	***0.987***	0.965	0.985	0.990	***0.981***	0.976	0.989	0.994	***0.987***
LiMM	0.949	0.967	0.981	***0.971***	0.960	0.981	0.988	***0.977***	0.931	0.968	0.983	***0.963***

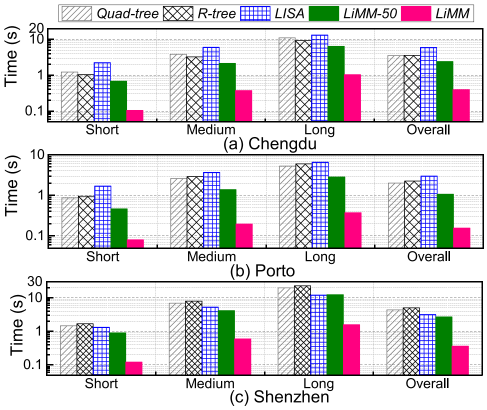


Fig. 9. Overall performance comparison on *matching time*.

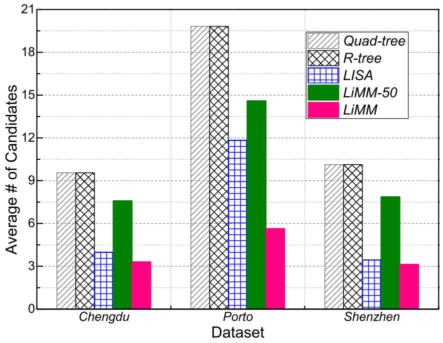


Fig. 10. Comparison on the average size of candidate sets.

TABLE V
COMPARISON ON THE *QUERY TIME* (UNIT: MS)

City	***Quad-tree***	***R-tree***	***LISA***	LiMM-***50***	LiMM
Chengdu	0.411	0.453	66.695	0.462	0.309
Porto	1.317	0.809	78.157	0.755	0.425
Shenzhen	1.746	0.512	67.296	0.583	0.409

(2) *Quad-tree* and *R-tree* derive the best matching accuracy across all experiments. With the same searching range $\gamma = 50\,\mathrm{m}$ as the two methods, LiMM-*50* achieves comparable matching accuracy, with an average gap of only 0.3%.

(3) The learned index *LISA* has the lowest matching accuracy. This is because *LISA* only returns the road segments, whose start and end vertices both fall into the searching circle, as the candidates, and thus will miss many "right" candidates, resulting in poor matching accuracy.

(4) With adaptive searching ranges, LiMM can achieve reasonably high accuracy. Compared to the best results, LiMM has only, on average, 1.9%, 0.6%, and 2.9% accuracy reduction on the *Chengdu*, *Porto*, and *Shenzhen* dataset, respectively. Overall, LiMM has comparable matching accuracy as *Quad-tree/R-tree*, with an average gap of only 1.8%.

Computation efficiency: We compare the *matching time* of all methods in Fig. 9, where logarithmic scale is applied to the y-axis for clear comparisons. Since the computation complexity of HMM model is highly related to the trajectory length, each method will take more time to map match a trajectory when it contains more GPS samples, just as shown in Fig. 9. Across all the experiments, our methods, both LiMM-*50* and LiMM, can complete the map matching tasks much faster than the other three methods. Moreover, LiMM further accelerates LiMM-*50* with a considerable speedup, e.g., on average enhancing the map matching process by $5.8\times$. According to the results in Fig. 9, LiMM significantly outperforms *Quad-tree*, *R-tree*, and *LISA* in terms of computation efficiency, with an average speedup of $10.3\times$, $11.4\times$, and $13.3\times$, respectively, across the three datasets.

Query processing: To better understand the results shown in Fig. 9, we further compare the average number of candidates returned by different indexing methods in Fig. 10. We observe that LiMM derives the smallest candidate sets, when compared to other methods, over the three road networks. Although *LISA* obtains the second smallest candidate sets, it needs to transform the query results back to specific road segments, introducing extra post-processing time. Therefore, the overall matching time of *LISA* is much greater than LiMM, as shown in Fig. 9. Even with the same searching range $\gamma = 50\,\mathrm{m}$ as *Quad-tree* and *R-tree*, LiMM-*50* retrieves much fewer candidates. It implies that the two traditional index structures are still redundant on the query results, while LiMM can precisely represent and index road segments, which benefits the map matching task to derive refined candidates.

Furthermore, we compare the query time of different methods in Table V. Specifically, *R-tree* is generally more efficient than *Quad-tree*, while *LISA* has the longest query time, due to the post-processing operations. LiMM is the most efficient on query processing. As LiMM can adaptively adjust the searching range, it performs better than LiMM-*50* and can return the candidate set for each GPS sample within $0.5\,\mathrm{m}s$.

Construction costs: Fig. 11 shows the index construction costs of all methods over the three road networks. Due to the varying distributions of road segments, each method consumes different time to construct the road network index, which may occupy different memory space. In terms of construction time,

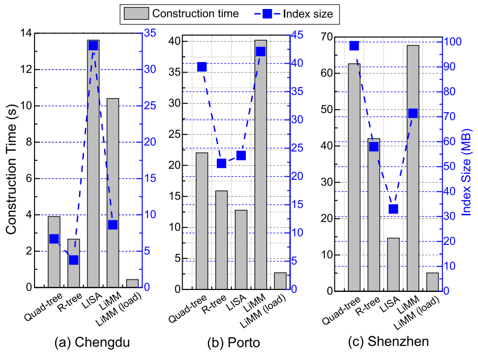


Fig. 11. Construction time and index sizes.

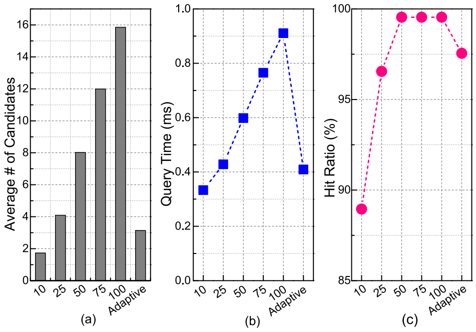


Fig. 12. Impact of different searching range γ values.

LiMM needs a bit more time than other methods, except on the *Chengdu*'s road network. While the index construction of LiMM is considerably prompt, as LiMM only takes $67.7s$ to build the index for *Shenzhen*'s road network, which is pretty large. In fact, we can save the pre-built index on the hard disk, and load it into memory within only a few seconds (i.e., $0.5s \sim 5s$), as shown by "LiMM *(load)*" in Fig. 11.

The index size of LiMM is comparable with traditional indexing methods, e.g., *Quad-tree*. The index size of LiMM is less than $75.0\,\mathrm{m}B$, which is negligible for modern machines.

C. Detailed Evaluations

Effectiveness of adaptive searching range γ: We conduct experiments to evaluate the query performance by comparing different γ values with adaptive searching range using the *Shenzhen* dataset. In principle, a larger γ generates a greater searching circle, which can find more candidate states (see Fig. 12(a)), leading to a longer query time (see Fig. 12(b)) and a higher hit ratio (see Fig. 12(c)). Instead of adopting a fixed γ value, LiMM adjusts the searching range for each area by analyzing historical trajectory data via RL models. Fig. 12 shows that our adaptive searching range can derive the most refined candidate sets, e.g., reducing irrelevant candidates up to 405% compared to $\gamma = 100\,\mathrm{m}$, yet with a moderate query time, i.e., $0.5\,\mathrm{m}s$, and a high hit ratio, i.e., 97.6%.

Effectiveness of coverage-oriented trajectory selections: To derive the adaptive searching range for each hexagon, LiMM needs to train a RL model using a set of map matched trajectories.

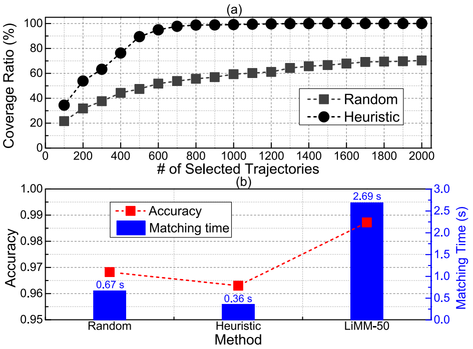


Fig. 13. Impact of different trajectory selection methods.

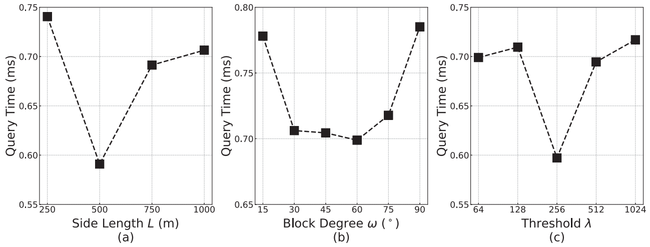


Fig. 14. Impact of different hexagon settings.

Rather than randomly selecting the training trajectories, we propose a heuristic method that aims to fully cover all hexagons. We conduct an experiment to compare the random and our heuristic method in the *Shenzhen* dataset, which has the most hexagons as 2249. In this experiment, we set the trajectory budget $\mathcal{B} = 2000$. As shown in Fig. 13(a), our heuristic method achieves 100% hexagon coverage using only 1400 trajectories, whereas the random method merely covers a maximum of 70.3% of the hexagons. In addition, our method covers each hexagon H_i multiple times, so that we can have sufficient samples to learn H_i's suitable searching range.

Fig. 13(b) compares the map matching performance of different trajectory selection methods. Compared to the fixed $\gamma = 50\,\mathrm{m}$ setting, both random and heuristic methods greatly improve the matching time by $3.0\times$ and $6.4\times$, respectively, at the accuracy loss of about 1.9% and 2.4%, respectively. If a hexagon is not covered by any trajectory samples, we cannot derive its adaptive searching range, and LiMM has to adopt the default $\gamma = 50\,\mathrm{m}$ setting. This is why random method has a bit higher accuracy than the heuristic method. With the learned adaptive searching range, heuristic method outperforms random method in terms of matching time with an improvement of 45.9%.

Impact of hexagon settings: The hexagon settings, including side length L, block degree threshold ω, and capacity threshold λ, determine the key assignments for road segments (see (4)), and thus affect LiMM's query performance, especially on the evaluation metric of query time.

We test various hexagon sizes by varying L. Fig. 14(a) shows that too small or too large hexagons will result in a longer query time. This is because a given searching circle may overlap with more hexagons if L is small, and LiMM has to verify many irrelevant candidates. On the other hand, large hexagons usually contain more candidates, which also increase the query time. According to our experiments, $L = 500\,\text{m}$ is a good choice for quick query processing.

We have observed similar experiment results for the block degree threshold ω, as shown in Fig. 14(b). A suitable ω should well distribute the road segment keys and facilitate query processing. We find that $\omega = 60^\circ$ is a good setting for our testing road networks.

Lastly, we explore the impact of capacity threshold λ. If a hexagon covers more than λ road segments, it is divided into sub-hexagons. Therefore, a small λ will generate many sub-hexagons and thus increase the query time. While a larger λ allows each hexagon to contain more road segments, and thus more irrelevant road segments are returned. Fig. 14(c) suggests that $\lambda = 256$ can lead to the smallest query time.

VII. Related Work

Map matching: Existing map matching works can be classified into two major categories, i.e., *traditional model-based approaches* and *emerging learning-based approaches*.

Traditional map matching models can be further divided into four classes [7], i.e., *similarity model*, *state-transition model*, *candidate evolving model*, and *scoring model*. Considering the wide popularity of Hidden Markov model (HMM) in map matching, we mainly discuss the HMM-based map matching approaches, while readers can refer to [7], [25], [29] for a comprehensive survey. As introduced in Section II-B, HMM consists of three stages for mapping a trajectory to its actual travel route [41]. Existing HMM-based approaches follow this workflow, and primarily focus on improving the matching accuracy by leveraging various information, e.g., road connectivity [32], [42], travel speeds [17], [21], moving directions [8], [10], [20], route choices [23], [49], [59], and driver behaviors [56]. Despite the high matching accuracy, HMM-based approaches still suffer from huge computation costs. Some research works make use of parallel-computing frameworks, e.g., MapReduce [22] and Spark [3], [58], or pre-computing techniques [54] to speedup the map matching process for large trajectory datasets. However, these methods require extra resources, e.g., clustered machines or memory storage.

Different from these works, `LiMM` builds a novel learned road network index and selects suitable searching ranges to optimize candidate preparations, which determine the computation complexity of HMM-based map matching approaches. Hence, `LiMM` provides refined candidate sets for existing approaches and thereby improves their efficiency.

Learning-based approaches exploit deep learning technique and abundant historical trajectory data to learn deep models that can directly transform a trajectory to its corresponding route [14], [26], [45], [47]. For example, based on the sequence-to-sequence (Seq2Seq) multi-task learning, *MTrajRec* [45] can recover low-sampling-rate trajectories and map match them to the road network simultaneously. *DeepMM* [14] exploits the Seq2Seq framework with attention mechanism to map trajectories to the road network. While *DMM* [47] utilizes a recurrent neural network to identify the most likely travel route for a sequence of cellular trajectory. Additionally, *L2MM* [26] employs multiple deep learning models to learn a mapping function from trajectories to travel routes. However, learning-based approaches heavily rely on massive well-labeled trajectories, and cannot tolerant road network changes.

Indexes for spatial data: Many traditional index structures can be employed to organize spatial data, and they are classified into three categories [16]: *(i) data partitioning based indexes*, e.g., *R-tree* [18] and *R*-tree* [4], *(ii) space partitioning based indexes*, e.g., *Quad-tree* [15], *Octree* [39] and *KD-tree* [5], and *(iii) mapping based indexes*, e.g., *B+ tree* [9] and *UB-tree* [44]. Existing HMM-based map matching approaches typically use *R-tree* or *Quad-tree* to index road networks.

Recently, Kraska et al. [28] introduce the idea of *learned index* to substitute traditional indexes with machine learning models. The intuition is to learn the cumulative distribution function (CDF) of a sorted dataset using implicit models that can effectively remember the storage addresses of all data. Some valuable learned index structures have been proposed to extend the idea of learned index to spatial data, including *ML-index* [11], *Tsunami* [13], *LISA* [30], *LHist* [31], *RSMI* [43], *Flood* [40], and *Qd-tree* [55]. Since the spatial data are generally multi-dimensional, these indexes thus first scale spatial data into one-dimensional values, and then learn the CDF of values with machine learning models [38]. For example, *ML-index* [11] utilizes the *iDistance* technique [24] to map data points to one-dimensional values and organizes them using multiple models. The state-of-the-art *LISA* [30] partitions data with grids and transforms them into one-dimensional values based on the numbered grid cells. The mapped values are further used to train models. Different from these works, we propose a learned index structure for the road networks by leveraging a novel hexagon-based scaling method, which particularly considers the characteristics of road segments.

Reinforcement learning (RL): RL aims to learn appropriate actions from observed states and received rewards based on the interactions between an agent and the environment [27]. Due to its generality, RL has been widely applied to solving various challenging problems, including vehicle dispatching [33], arrangement of crowdsourcing tasks [46], network congestion control [53], APP usage prediction [48], distance querying in road networks [61], [62], and graph matching [50], [51]. In this work, we employ RL to determine suitable searching ranges for HMM-based map matching, which can greatly reduce the subsequent computations.

VIII. Conclusion

In this paper, we present `LiMM` to alleviate the computation bottleneck of HMM-based map matching. We design `LiMM` with two functional modules, i.e., a learned road network index and an RL-based adaptive searching range mechanism. The two key designs together refine candidate states for existing HMM-based map matching approaches, and thus improve their computation efficiency. Extensive experiments are performed with three large real-world trajectory datasets, and the results demonstrate that

LiMM significantly outperforms baseline methods with an average speedup of 11.7× in terms of matching time, yet with a subtle accuracy loss of only 1.8%.

References

[1] Openstreetmap. Accessed: Jun. 2024. [Online]. Available: http://www.openstreetmap.org/

[2] Porto dataset in kaggle. Accessed: Jun. 2024. [Online]. Available: https://www.kaggle.com/competitions/pkdd-15-predict-taxi-service-trajectory-i/data

[3] D. Alves Peixoto, H. Quoc Viet Nguyen, B. Zheng, and X. Zhou, “A framework for parallel map-matching at scale using Spark,” *Distrib. Parallel Databases*, vol. 37, no. 4, pp. 697–720, 2019.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, “The R*-tree: An efficient and robust access method for points and rectangles,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1990, pp. 322–331.

[5] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[6] C. Cao, Z. Liu, M. Li, W. Wang, and Z. Qin, “Walkway discovery from large scale crowdsensing,” in *Proc. 17th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw.*, 2018, pp. 13–24.

[7] P. Chao, Y. Xu, W. Hua, and X. Zhou, “A survey on map-matching algorithms,” in *Proc. Australas. Database Conf.*, Springer, 2020, pp. 121–133.

[8] C. Chen, Y. Ding, X. Xie, S. Zhang, Z. Wang, and L. Feng, “TrajCompressor: An online map-matching-based trajectory compression framework leveraging vehicle heading direction and change,” *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 5, pp. 2012–2028, May 2020.

[9] D. Comer, “Ubiquitous B-tree,” *ACM Comput. Surv.*, vol. 11, no. 2, pp. 121–137, 1979.

[10] G. Cui, W. Bian, and X. Wang, “Hidden markov map matching based on trajectory segmentation with heading homogeneity,” *GeoInformatica*, vol. 25, pp. 179–206, 2021.

[11] A. Davitkova, E. Milchevski, and S. Michel, “The ML-Index: A multidimensional, learned index for point, range, and nearest-neighbor queries,” in *Proc. 23rd Int. Conf. Extending Database Technol.*, 2020, pp. 407–410.

[12] J. Ding et al., “ALEX: An updatable adaptive learned index,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 969–984.

[13] J. Ding, V. Nathan, M. Alizadeh, and T. Kraska, “Tsunami: A learned multi-dimensional index for correlated data and skewed workloads,” in *Proc. VLDB Endowment*, vol. 14, no. 2, pp. 74–86, 2020.

[14] J. Feng et al., “DeepMM: Deep learning based map matching with data augmentation,” *IEEE Trans. Mobile Comput.*, vol. 21, no. 7, pp. 2372–2384, Jul. 2022.

[15] R. A. Finkel and J. L. Bentley, “Quad trees: A data structure for retrieval on composite keys,” *Acta Informatica*, vol. 4, no. 1, pp. 1–9, 1974.

[16] V. Gaede and O. Günther, “Multidimensional access methods,” *ACM Comput. Surv.*, vol. 30, no. 2, pp. 170–231, 1998.

[17] C. Y. Goh, J. Dauwels, N. Mitrovic, M. T. Asif, A. Oran, and P. Jaillet, “Online map-matching based on hidden markov model for real-time traffic sensing applications,” in *Proc. IEEE Conf. Intell. Transp. Syst.*, 2012, pp. 776–781.

[18] A. Guttman, “R-trees: A dynamic index structure for spatial searching,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1984, pp. 47–57.

[19] A. Hansson, E. Korsberg, R. Maghsood, and E. Nordén, and Selpi, “Lane-level map matching based on HMM,” *IEEE Trans. Intell. Veh.*, vol. 6, no. 3, pp. 430–439, Sep. 2021.

[20] Y.-L. Hsueh and H.-C. Chen, “Map matching for low-sampling-rate GPS trajectories by exploring real-time moving directions,” *Inf. Sci.*, vol. 433, pp. 55–69, 2018.

[21] G. Hu, J. Shao, F. Liu, Y. Wang, and H. T. Shen, “If-matching: Towards accurate map-matching with information fusion,” *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 1, pp. 114–127, Jan. 2017.

[22] J. Huang, S. Qiao, H. Yu, J. Qie, and C. Liu, “Parallel map matching on massive vehicle GPS data using MapReduce,” in *Proc. 10th IEEE Int. Conf. High Perform. Comput. Commun. IEEE Int. Conf. Embedded Ubiquitous Comput.*, 2013, pp. 1498–1503.

[23] G. R. Jagadeesh and T. Srikanthan, “Online map-matching of noisy and sparse location data with hidden markov and route choice models,” *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 9, pp. 2423–2434, Sep. 2017.

[24] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang, “iDistance: An adaptive B-tree based indexing method for nearest neighbor search,” *ACM Trans. Database Syst.*, vol. 30, no. 2, pp. 364–397, 2005.

[25] L. Jiang, C. Chen, C. Chen, H. Huang, and B. Guo, “From driving trajectories to driving paths: A survey on map-matching algorithms,” *CCF Trans. Pervasive Comput. Interaction*, vol. 4, no. 3, pp. 252–267, 2022.

[26] L. Jiang, C.-X. Chen, and C. Chen, “L2MM: Learning to map matching with deep models for low-quality GPS trajectory data,” *ACM Trans. Knowl. Discov. Data*, vol. 17, no. 3, pp. 1–25, 2023.

[27] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, 1996.

[28] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, “The case for learned index structures,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2018, pp. 489–504.

[29] M. Kubicka, A. Cela, H. Mounier, and S.-I. Niculescu, “Comparative study and application-oriented classification of vehicular map-matching methods,” *IEEE Intell. Transp. Syst. Mag.*, vol. 10, no. 2, pp. 150–166, Summer, 2018.

[30] P. Li, H. Lu, Q. Zheng, L. Yang, and G. Pan, “LISA: A learned index structure for spatial data,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 2119–2133.

[31] Q. Liu, Y. Shen, and L. Chen, “LHist: Towards learning multi-dimensional histogram for massive spatial data,” in *Proc. IEEE Int. Conf. Data Eng.*, 2021, pp. 1188–1199.

[32] X. Liu, K. Liu, M. Li, and F. Lu, “A ST-CRF map-matching method for low-frequency floating car data,” *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 5, pp. 1241–1254, May, 2017.

[33] Z. Liu, J. Li, and K. Wu, “Context-aware taxi dispatching at city-scale using deep reinforcement learning,” *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 3, pp. 1996–2009, Mar. 2022.

[34] Z. Liu, Z. Li, M. Li, W. Xing, and D. Lu, “Mining road network correlation for traffic estimation via compressive sensing,” *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 7, pp. 1880–1893, Jul. 2016.

[35] Z. Liu, Z. Li, K. Wu, and M. Li, “Urban traffic prediction from mobility data using deep learning,” *IEEE Netw.*, vol. 32, no. 4, pp. 40–46, Jul./Aug. 2018.

[36] Z. Liu, J. Liu, X. Xu, and K. Wu, “DeepGPS: Deep learning enhanced GPS positioning in urban canyons,” *IEEE Trans. Mobile Comput.*, vol. 23, no. 1, pp. 376–392, Jan. 2024.

[37] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang, “Map-matching for low-sampling-rate GPS trajectories,” in *Proc. 17th ACM SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, 2009, pp. 352–361.

[38] R. Marcus et al., “Benchmarking learned indexes,” in *Proc. VLDB Endowment*, vol. 14, no. 1, pp. 1–13, 2020.

[39] D. Meagher, “Geometric modeling using octree encoding,” *Comput. Graph. Image Process.*, vol. 19, no. 2, pp. 129–147, 1982.

[40] V. Nathan, J. Ding, M. Alizadeh, and T. Kraska, “Learning multi-dimensional indexes,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 985–1000.

[41] P. Newson and J. Krumm, “Hidden markov map matching through noise and sparseness,” in *Proc. ACM SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, 2009, pp. 336–343.

[42] T. Osogami and R. Raymond, “Map matching with inverse reinforcement learning,” in *Proc. Int. Joint Conf. Artif. Intell.*, 2013, pp. 1–7.

[43] J. Qi, G. Liu, C. S. Jensen, and L. Kulik, “Effectively learning spatial indices. *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 2341–2354, 2020.

[44] F. Ramsak, V. Markl, R. Fenk, M. Zirkel, K. Elhardt, and R. Bayer, “Integrating the UB-tree into a database system kernel,” in *Proc. 26th Int. Conf. Very Large Data Bases*, 2000, pp. 263–272.

[45] H. Ren et al., “MTrajRec: Map-constrained trajectory recovery via Seq2Seq multi-task learning,” in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2021, pp. 1410–1419.

[46] C. Shan, N. Mamoulis, R. Cheng, G. Li, X. Li, and Y. Qian, “An end-to-end deep RL framework for task arrangement in crowdsourcing platforms,” in *Proc. IEEE Int. Conf. Data Eng.*, 2020, pp. 49–60.

[47] Z. Shen, W. Du, X. Zhao, and J. Zou, “DMM: Fast map matching for cellular data,” in *Proc. ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2020, pp. 1–14.

[48] Z. Shen, K. Yang, W. Du, X. Zhao, and J. Zou, “DeepAPP: A deep reinforcement learning framework for mobile application usage prediction,” in *Proc. 17th Conf. Embedded Netw. Sensor Syst.*, 2019, pp. 153–165.

[49] S. Taguchi, S. Koide, and T. Yoshimura, “Online map matching with route prediction,” *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 1, pp. 338–347, Jan. 2019.

[50] H. Wang, Y. Zhang, L. Qin, W. Wang, W. Zhang, and X. Lin, “Reinforcement learning based query vertex ordering model for subgraph matching,” in *Proc. IEEE Int. Conf. Data Eng.*, 2022, pp. 245–258.

[51] Y. Wang, Y. Tong, C. Long, P. Xu, K. Xu, and W. Lv, "Adaptive dynamic bipartite graph matching: A reinforcement learning approach," in *Proc. IEEE Int. Conf. Data Eng.*, 2019, pp. 1478–1489.

[52] H. Wu, W. Sun, and B. Zheng, "Is only one GPS position sufficient to locate you to the road network accurately?," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2016, pp. 740–751.

[53] R. Xie, X. Jia, and K. Wu, "Adaptive online decision method for initial congestion window in 5G mobile edge computing using deep reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 389–403, Feb. 2020.

[54] C. Yang and G. Gidofalvi, "Fast map matching, an algorithm integrating hidden markov model with precomputation," *Int. J. Geographical Inf. Sci.*, vol. 32, no. 3, pp. 547–570, 2018.

[55] Z. Yang et al., "QD-Tree: Learning data layouts for Big Data analytics," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 193–208.

[56] Y. Yin, R. R. Shah, G. Wang, and R. Zimmermann, "Feature-based map matching for low-sampling-rate GPS trajectories," *ACM Trans. Spatial Algorithms Syst.*, vol. 4, no. 2, pp. 1–24, 2018.

[57] H. Yuan, G. Li, Z. Bao, and L. Feng, "Effective travel time estimation: When historical trajectories over road networks matter," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 2135–2149.

[58] A. Zeidan, E. Lagerspetz, K. Zhao, P. Nurmi, S. Tarkoma, and H. T. Vo, "GeoMatch: Efficient large-scale map matching on apache spark," *ACM Trans. Data Sci.*, vol. 1, no. 3, pp. 1–30, 2020.

[59] Y. Zhang and X. Sui, "RCIVMM: A route choice-based interactive voting map matching approach for complex urban road networks," *IEEE Trans. Big Data*, vol. 8, no. 5, pp. 1414–1427, Oct. 2022.

[60] B. Zheng et al., "SpeakNav: Voice-based route description language understanding for template-driven path search," in *Proc. VLDB Endowment*, vol. 14, no. 12, pp. 3056–3068, 2021.

[61] B. Zheng et al., "Reinforcement learning based tree decomposition for distance querying in road networks," in *Proc. IEEE Int. Conf. Data Eng.*, 2023, pp. 1678–1690.

[62] B. Zheng et al., "Workload-aware shortest path distance querying in road networks," in *Proc. IEEE Int. Conf. Data Eng.*, 2022, pp. 2372–2384.

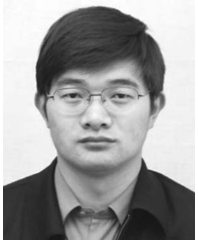
[63] Y. Zheng, "Trajectory data mining: An overview," *ACM Trans. Intell. Syst. Technol.*, vol. 6, no. 3, pp. 1–41, 2015.

Zhidan Liu (Member, IEEE) received the PhD degree in computer science and technology from Zhejiang University, Hangzhou, China, in 2014. After that, he worked as a research fellow with Nanyang Technological University, Singapore, and a faculty member with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. He is currently an assistant professor with Intelligent Transportation Thrust, System Hub, The Hong Kong University of Science and Technology (Guangzhou). His research interests include Internet of Things, urban computing, and Big Data analytic. He is a senior member of CCF and a member of ACM.

Yingqian Zhou received the BS degree from the Guangdong University of Technology, Guangzhou, China, in 2020, and the master's degree from Shenzhen University, Shenzhen, China, under the supervision of Dr. Zhidan Liu, in 2023. Her research interests are in the areas of trajectory data analysis and urban computing.

Xiaosi Liu received the BE degree in software engineering from Nanchang University, Nanchang, China, in 2022. She is currently working toward the master's degree with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China, under the supervision of Dr. Zhidan Liu. Her research interests include spatio-temporal data analysis and urban computing.

Haodi Zhang (Member, IEEE) received the PhD degree from the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, in 2016. He is currently a tenured associate professor with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. His research interests include knowledge representation and reasoning, deep learning, crowdsourcing over probabilistic and uncertain databases, and natural language processing



Yabo Dong received the PhD degree in computer science from Zhejiang University, Hangzhou, China, in 2002. He is currently an associate professor with the College of Computer Science and Technology, Zhejiang University. His research interests include Internet of Things, sensor data mining and processing, and cultural relics protection.

Dongming Lu is currently a professor with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China, and the vice president with NingboTech University. He was selected as a member of the Program for New Century Excellent Talents by the Ministry of Education of China, in 2004, and one of the 151 Talents Program from Zhejiang Province, in 2008. He has co-authored more than 200 refereed papers and has 34 patents granted. His research interests include virtual reality, computer vision, Internet of Things, and Big Data.

Kaishun Wu (Fellow, IEEE) received the PhD degree in computer science and engineering with The Hong Kong University of Science and Technology. Before joining HKUST(GZ) as a full professor with DSA Thrust and IoT Thrust, in 2022, he was a distinguished professor and director with Guangdong Provincial Wireless Big Data and Future Network Engineering Center, Shenzhen University. He is an active researcher with more than 200 papers published on major international academic journals and conferences, as well as more than 100 invention patents, including 12 from the USA. He received the 2012 Hong Kong Young Scientist Award, the 2014 Hong Kong ICT Awards: Best Innovation, and 2014 IEEE ComSoc Asia-Pacific Outstanding Young researcher Award. He is an fellow of IET, and AAIA.