mathematics

MDPI

Article

FurMoLi: A Future Query Technique for Moving Objects Based on a Learned Index

Jiwei Yang [1], Chong Zhang [1,*], Wen Tang [1], Bin Ge [1], Hongbin Huang [1] and Shiyu Yang [2]

[1] Laboratory for Big Data and Decision, National University of Defense Technology, Changsha 410000, China; yangjiwei18@nudt.edu.cn (J.Y.); stangwa@163.com (W.T.); gebin@nudt.edu.cn (B.G.); hbhuang@nudt.edu.cn (H.H.)
[2] Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou 510006, China; shiyu.yang@aliyun.com
* Correspondence: chongzhang@nudt.edu.cn

Abstract: The future query of moving objects involves predicting their future positions based on their current locations and velocities to determine whether they will appear in a specified area. This technique is crucial for positioning and navigation, and its importance in our daily lives has become increasingly evident in recent years. Nonetheless, the growing volume of data renders traditional index structures for moving objects, such as the time-parameterized R-tree (TPR-tree), inefficient due to their substantial storage overhead and high query costs. Recent advancements in learned indexes have demonstrated a capacity to significantly reduce storage overhead and enhance query efficiency. However, most existing research primarily addresses static data, leaving a gap in the context of future queries for moving objects. We propose a novel **f**uture q**uer**y technique for **m**oving **o**bjects based on a **l**earned **i**ndex (**FurMoLi** for short). FurMoLi encompasses four key stages: firstly, a data partition through clustering based on velocity and position information; secondly, a dimensionality reduction mapping two-dimensional data to one dimension; thirdly, the construction of a learned index utilizing piecewise regression functions; and finally, the execution of a future range query and future KNN query leveraging the established learned index. The experimental results demonstrate that FurMoLi requires 4 orders of magnitude less storage overhead than TPR-tree and 5 orders of magnitude less than B^+-tree for moving objects (B^x-tree). Additionally, the future range query time is reduced to just 41.6% of that for TPR-tree and 34.7% of that for B^x-tree. For future KNN queries, FurMoLi's query time is only 70.1% of that for TPR-tree and 47.4% of that for B^x-tree.

Keywords: moving objects; machine learning; learned index; clustering; future range query; future KNN query

MSC: 68T02

check for updates

Citation: Yang, J.; Zhang, C.; Tang, W.; Ge, B.; Huang, H.; Yang, S. FurMoLi: A Future Query Technique for Moving Objects Based on a Learned Index. *Mathematics* **2024**, *12*, 2032. https://doi.org/10.3390/math12132032

Academic Editor: Ioannis G. Tsoulos

Received: 27 May 2024
Revised: 22 June 2024
Accepted: 28 June 2024
Published: 29 June 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Future queries oriented toward moving objects are pivotal in various real-world applications. For instance, in path planning and navigation, executing a future range query on a map reveals the projected locations of moving vehicles within a specified area. This information aids drivers in circumventing traffic congestion and selecting the optimal route. Additionally, this technology is extremely valuable in fields such as logistics management, traffic control, and emergency response, including safety and disaster relief efforts [1,2]. Therefore, decades of research by scholars worldwide has focused on developing specialized index structures for moving objects. These include TPR-tree [3] and its advanced version, TPR*-tree [4], which are primarily based on the conventional spatial index structure R-tree [5] or its derivative, the R*-tree [6]. These index structures efficiently prune irrelevant data during queries and are highly adaptive to the dynamics

of moving objects. However, in the era of big data, these traditional index structures face significant challenges.

The relentless growth in data scale necessitates expansive index structures, which can sometimes surpass the size of the underlying datasets themselves. Research indicates that, in today's commercial databases, index structures can occupy as much as 55% of server memory [7]. As data volumes expand, it becomes impractical for main memory to house the entire index structure, necessitating partial offloading to external memory [8]. This imposes considerable storage pressures on databases and leads to increased I/O operations. The hierarchical nature of these index structures, requiring access from root to leaf nodes, exacerbates the situation. As the volume of data swells, so does the number of tree nodes that must be accessed during queries, significantly slowing down query performance. This issue highlights a critical weakness in traditional R-tree-based index structures in the era of big data. Given these deficiencies, there is a critical need to innovate the index structures for moving objects, ensuring they can cope with the increasing data scales and the stringent demands for query efficiency.

In 2018, Kraska et al. [9] introduced the concept of a learned index, utilizing a Recursive Model Index (RMI) as an alternative to traditional B-trees. This index employs a multi-layer machine learning model that learns the distribution patterns of the underlying data, effectively creating a mapping similar to the cumulative distribution function (CDF) of approximate keys. As illustrated in Figure 1, for a given query key x, the RMI predicts the position of x by learning $\mathrm{F}(x)$ from the CDF. The exact location of the key is then determined within the interval $[\mathrm{F}(x) - \varepsilon, \mathrm{F}(x) + \varepsilon]$, where ε represents the permissible error bound.

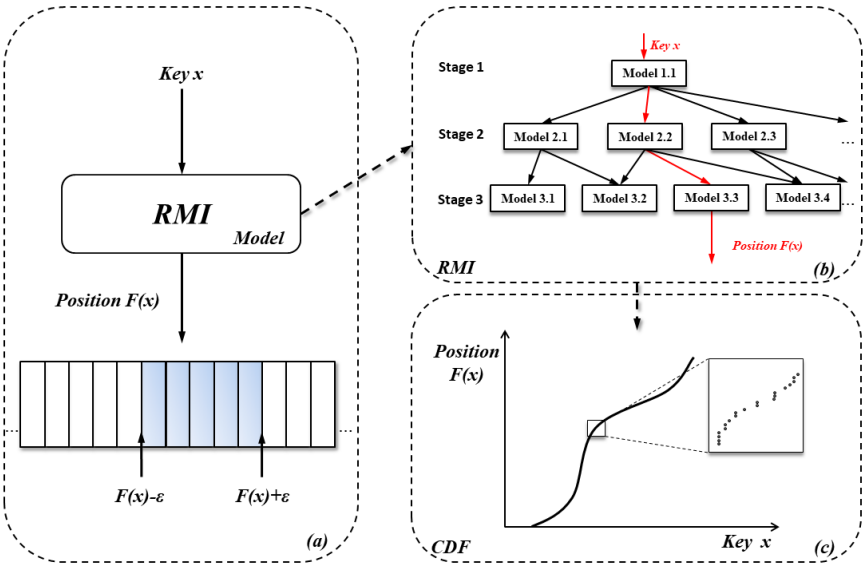


Figure 1. RMI diagram. (**a**) Framework of RMI. (**b**) Staged models. (**c**) Index as CDF.

A learned index is an innovative index structure that is constructed by leveraging patterns in the data distribution, thereby exhibiting high adaptability to varying data. Unlike traditional tree structures, which rely on traversal searches, a learned index performs queries through direct computations, significantly enhancing query efficiency. Additionally, since it stores only the parameters of its models—which are independent of the size of the underlying dataset—it also offers advantages in reducing memory overhead compared to traditional index structures. A series of one-dimensional learned indexes such as the RMI have demonstrated limitations when applied to high-dimensional data [10]. In response, various high-dimensional learned indexes have been developed in the past three years, including LISA [10], Flood [11], the RSMI [12], and the ZM Index [13]. However, these studies predominantly focus on static data. Although there have been advancements in updating methods for high-dimensional learned indexes, their application scenarios remain

confined to current or historical data queries [14]. Notably, the existing spatial learning indexes are not equipped to support future-oriented queries for moving objects.

This paper proposes **FurMoLi**, which employs a machine learning model to construct an index for supporting future queries of moving objects through a series of meticulously designed steps. Recognizing the dynamic nature of moving objects, FurMoLi adopts the clustering approach by Li et al. [15], which accounts for both location and velocity attributes. This method clusters the dataset into K regions, ensuring that each region maintains similar positional and motion characteristics to the extent possible. During the dimensionality reduction phase, we sort and number the cluster centers using the Lebesgue measure [16] and create a partially monotonic dimensionality reduction function based on the partition boundaries. This function maps and sorts the two-dimensional data points of moving objects into a one-dimensional space, preserving the original order of dimensionality reduction values within each region and preventing cross-region value overlaps. In the model training phase, FurMoLi learns a series of piecewise linear functions, optimizing space efficiency, and employs the least squares method to fine-tune the parameters of each function. In the future query phase, FurMoLi draws on the mechanism of B^x-tree [17] to expand the query windows, facilitating future range and KNN queries for moving objects.

The main contributions of this paper are as follows:

(1) A novel learned index structure and an effective query algorithm based on the learned index structure are designed to support future queries of moving objects, including future range queries and future k-nearest neighbor (KNN) queries. As far as we know, this is the first mature learned index for future queries of moving objects.

(2) A clustering method that considers both location and velocity attributes is utilized to segment the data region, ensuring uniform motion states within each region. This approach enhances the model's capability to effectively learn the data distribution. Meanwhile, on this basis, a creative dimension reduction method is adopted to ensure the query recall rate and greatly reduce the calculation difficulty. Just add a simple filtering operation to ensure the accuracy of the query and improve the query efficiency.

(3) Extensive experiments are conducted on synthetic datasets. The results show that FurMoLi has better storage overhead and query efficiency than traditional moving object index structures, while maintaining good construction efficiency.

In Section 2, we discuss technologies relevant to this paper. Section 3 details the index construction process for moving objects and elaborates on future queries. The experimental results and analysis are presented in Section 4, with conclusions and future works provided in Section 5.

2. Related Work

In this section, we review some methods related to FurMoLi. Section 2.1 is the introduction of traditional moving object index structures, and Section 2.2 covers multidimensional learned indexes.

2.1. Traditional Index Structures for Future Queries of Moving Objects

A moving object is defined as a spatial object whose position continuously changes over time, which can be either continuous or discrete [18]. The query technology for moving objects is categorized into future information queries and historical information queries [19]. This paper primarily focuses on the index structure for future information queries.

Based on R*-tree, Saltenis et al. [3] proposed TPR-tree, which employs the time–distance formula $x_t = x_0 + v \cdot \Delta t$ for uniform linear motion. This formula is used to time-parameterize data points of moving objects and their minimum bounding rectangles (MBRs), allowing the MBRs to evolve continuously with the moving objects they encompass. Future queries on moving objects are conducted by assessing the intersection between the query rectangle and the expanded MBR. However, the TPR-tree does not provide a means to quantify and optimize query performance. Tao et al. [4] introduced TPR*-tree, which considers the overlap of future MBRs. This enhanced version features a

new insertion strategy and an improved splitting algorithm. Additionally, it optimizes the dynamic adjustment strategy for MBR boundaries, quantifies query performance, and thereby enhances both query efficiency and index performance. In response to the observed degradation in TPR-tree query performance over time, Procopiuc et al. [20] introduced the concept of automatic adjustment and proposed the Spatio-Temporal Self-Adjusting R-tree (STAR-tree). This innovative structure allows for automatic adjustments whenever there is a decline in query performance. Tao et al. [21] proposed the Spatio-Temporal Prediction tree (STP-tree), designed to handle moving objects with unknown motion patterns. This approach involves describing the motion of moving objects using different trajectory functions on both the server and client sides. This method facilitates the creation of an expected trajectory index. Jensen et al. [17] proposed the B^x-tree based on the B^+-tree and employed a space-filling curve to map and order discrete moving points in multidimensional space, ensuring that points adjacent in the spatial domain are also adjacent in one-dimensional space, using the B^+-tree for the index, which provides high query and update performance. Different from TPR-tree, B^x-tree implements future queries on moving objects by enlarging the query rectangle. Yiu et al. [22] utilized the Hilbert curve to index moving objects in dual space and proposed the Indexing Moving Objects by Space Filling Curves in Dual Space (B^{dual}-tree), which is based on the B^+-tree structure. This approach helps to minimize the occurrence of incorrect results in future queries. The traditional index structures can effectively realize the future queries of moving objects; however, as the scale of moving objects and application demands grows, these traditional techniques often experience a decline in query performance. Additionally, the disadvantage of excessive index storage overhead becomes increasingly apparent. These traditional index structures of moving objects are summarized in Table 1.

Table 1. Summary of traditional index structures for moving objects.

Index Structure	Underlying Structure	Dimensionality Reduction Method	Characteristic	Representation of Moving Objects
TPR-tree	R^*-tree	/	Supports future query of moving objects	Parameterized MBR
TPR*-tree	R^*-tree	/	Optimizes the dynamic adjustment strategy for MBR	Parameterized MBR
STAR-tree	R^*-tree	/	Automatic adjustments	Parameterized MBR
STP-tree	R^*-tree	/	Supports unknown motion patterns	Segment
B^x-tree	B^+-tree	Space-filling curve	Enlarges the query rectangle	2-dimensional points
B^{dual}-tree	B^+-tree	Hilbert curve	Indexes in the dual space	Parameterized MBR

2.2. Multidimensional Learned Index

Since most multidimensional data are disorganized, it is relatively difficult to learn the cumulative distribution function of the multidimensional data directly using the RMI model for the machine learning model that more easily learns the distribution characteristics of ordered data. The concept of the multidimensional learned index was first proposed in the literature [23]. Currently, the mainstream multidimensional learned index is mainly divided into dimensionality-reduction-based methods and non-dimensionality-reduction-based methods.

Wang et al. [13] proposed the Z-order Model (ZM) index, which utilizes the Z-order curve to reduce the dimensionality of multidimensional data, transforming it into an ordered one-dimensional Z-address to preserve the original spatial relationships of the data.

Additionally, a multi-layer model similar to the RMI is employed to index the Z-address, enabling the transformation of multidimensional queries into one-dimensional queries. The Recursive Spatial Model Index (RSMI) index proposed by Qi et al. [12] initially maps the original data to a grid-sorted space before employing a space-filling curve for dimensionality reduction and sorting. This method employs multi-layer perceptrons to learn the one-dimensional sorting of multidimensional data, so as to optimize the disadvantages of the uneven distribution of the ZM index multidimensional data after conversion. Li et al. [10] introduced a Learned Index Structure for Spatial Data (LISA), a multidimensional learned index that segments the data space into cells and applies the Lebesgue measure for dimensionality reduction. To decrease the model's complexity, the space is partitioned into multiple shards using piecewise linear functions. Additionally, a local model is developed to assign these shards to disk pages, facilitating efficient data retrieval. During queries, the addresses of the pages containing the relevant shards are returned, followed by a precise search within those shards. The multidimensional learned (ML) index was proposed by Davitkova et al. [24], which adopts a method similar to the iDistance index [25] for dimensionality reduction. Subsequently, the one-dimensional value of the data is indexed using the RMI model, enabling efficient data retrieval and query performance. Zhang et al. [26] introduced an efficient SPatial inteRpolation functIon based Grid index (SPRIG), a learned index that employs bilinear interpolation functions to learn the distribution of multidimensional data. This index is constructed in a non-dimensionality reduction manner, allowing for the direct handling of multidimensional data without transforming it into lower-dimensional representations. Nathan et al. [11] proposed A Multidimensional In-Memory Read-Optimized Index, Flood, a non-dimensionality reduction multidimensional learned index that dynamically adjusts its structure to accommodate varying data distributions and query demands. Unlike other multidimensional learned indexes that primarily focus on data distribution characteristics, Flood comprehensively addresses both the data distribution and the specific requirements of queries. A summary of these learned indexes is shown in Table 2.

In their experiments, researchers demonstrated that multidimensional learned indexes outperforms traditional multidimensional indexes in terms of query efficiency and space utilization. However, these methods were limited to static multidimensional data. This paper represents the first approach to applying a learned index to the future query of moving objects.

Table 2. Summary of multidimensional learned indexes.

Index Structure	Learning Model	Dimensionality Reduction Method	Advantages	Disadvantages
ZM index	Recursive model	Z-order curve	Small storage overhead	Updates and KNN query are not supported
RSMI	Recursive linear regression model	Space-filling curve	Supports update	High maintenance costs
LISA	Piecewise linear regression model	Lebesgue measure	Supports update and disk operations	High maintenance costs
ML index	Regression model	iDistance	No overlapping partitions	Update is not supported
SPRIG	Spatial interpolation function	/	High KNN query efficiency	Update is not supported

Table 2. *Cont.*

Index Structure	Learning Model	Dimensionality Reduction Method	Advantages	Disadvantages
Flood	Piecewise linear regression model	/	Adapt to different data distribution	Update is not supported

3. The Design of FurMoLi

This section will give an in-depth introduction to the design and training ideas of FurMoLi. Its overall architecture is shown in Figure 2. The overall process includes partitioning data areas, constructing functions, building piecewise linear learned indexes, and implementing future queries. Firstly, the "position–velocity" clustering method is employed to segment the initial moving object dataset into K regions based on the position and motion states of the data. These regions are then numbered, and a dimensionality reduction function is constructed to map two-dimensional moving object data into one-dimensional space and order them. To better understand the distribution of moving object data, a linear regression model is developed for each region, collectively forming a piecewise linear learned index. Finally, by expanding the query window, the dimensionality reduction values at the intersections between the expanded query window and the regions are calculated. This process determines the scope of the expanded query window on the one-dimensional index, enabling the retrieval of the required query points.

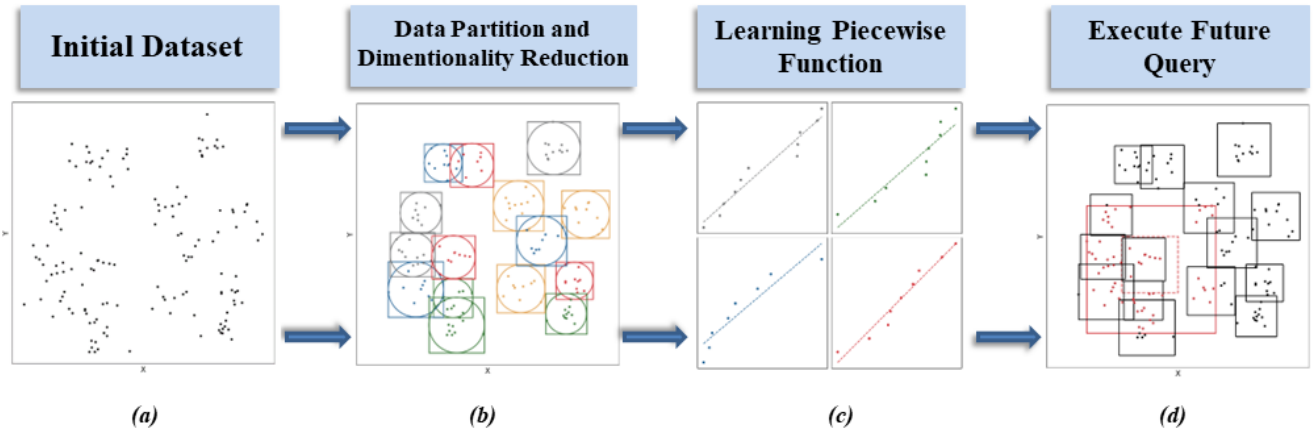


Figure 2. Flow diagram of FurMoLi. (**a**) Initial dataset. (**b**) Data partition and dimensionality reduction. (**c**) Build a segmented learned index. (**d**) Execute future queries. The points in red are the query points.

3.1. Data Partition

In the real world, the distribution of moving objects is often complicated. Without preprocessing and segmenting the data, directly learning from the dataset tends to result in low accuracy of the trained learned index and higher training costs. To more effectively learn the data distribution and facilitate subsequent operations, we fully utilized the location and velocity attributes of moving object data. We adopted the "location–velocity" clustering method, which is a variant of the K-Means clustering algorithm [27]. This variant modifies the distance calculation formula to include the influence of the moving objects' speed information. Consider two moving objects in space, $O_1(oid_1, x_1, y_1, vx_1, vy_1)$ and $O_2(oid_2, x_2, y_2, vx_2, vy_2)$, the clustering distance between O_1 and O_2 is defined as Equation (1) to account for both their spatial coordinates and velocities.

$$\mathrm{dist}^2(O_1, O_2) = (x_1 - x_2)^2 + (y_1 - y_2)^2 + \alpha(v_{x1} - v_{x2})^2 + \alpha\left(v_{y1} - v_{y2}\right)^2 \quad (1)$$

where α is the weight assigned to the velocity information. Given that the velocity attribute significantly impacts clustering distance calculations for moving objects, typically set $\alpha > 1$.

The specific process of dividing the data space is detailed below and outlined in Algorithm 1. First, K data points from the moving objects are randomly selected as the centroids. For each remaining data point $X_i(oid_i, x_i, y_i, vx_i, vy_i)$ and for each centroid $O_j(x_j, y_j, vx_j, vy_j)$ (where $j = 1, 2, \ldots, K$), the distance is calculated using Equation (1). This calculation helps determine the region to which each data point belongs by minimizing the distance between the data point and the centroid of its region, as expressed in Equation (2).

$$C^{(i)} = \arg\min_{j} \left(\mathrm{dist}^2 \left(X_i, O_j \right) \right) \tag{2}$$

where $C^{(i)}$ represents the region of the moving object X_i. The centroids' positions and velocities are then updated in Equation (3) to iterate the process until the model converges (when the change in centroid Δ exceeds the predefined threshold) or the maximum number of iterations is reached. The change in centroid Δ is defined by Equation (4).

$$\begin{cases} x'_{oj} = \dfrac{1}{|C_j|} \displaystyle\sum_{x \in C_j} x \\ y'_{oj} = \dfrac{1}{|C_j|} \displaystyle\sum_{y \in C_j} y \\ v'_{xoj} = \dfrac{1}{|C_j|} \displaystyle\sum_{v_x \in C_j} v_x \\ v'_{yoj} = \dfrac{1}{|C_j|} \displaystyle\sum_{v_y \in C_j} v_y \end{cases} \tag{3}$$

$$\Delta = \sum_{K} \left(\mathrm{dist}^2 \left(O_j, O'_j \right) \right) \tag{4}$$

where C_j refers to region j, $|C_j|$ refers to the total number of moving objects in region j, and $O'_j(x'_{oj}, y'_{oj}, v'_{xoj}, v'_{yoj})$ refers to the updated centroid of region j. Additionally, since calculating the intersection of rectangles is simpler than that of rectangles and circles, it is practical for the subsequent operations of FurMoLi to construct a minimum covering circle for each region using the finally determined centroid coordinates as the center. Subsequently, the minimum circumscribed rectangle of the circle is constructed. This rectangle is then used as the final boundary for the data division of the moving objects, ensuring the ease of calculation and integration into the FurMoLi system. Equations (5)–(7) build the minimum covering circle and the minimum circumscribed rectangle for each region.

$$r_j^2 = \max_{X_i \in C_j} \left(\left(x_i - x'_{oj} \right)^2 + \left(y_i - y'_{oj} \right)^2 \right) \tag{5}$$

$$\left(x - x'_{oj} \right)^2 + \left(y - y'_{oj} \right)^2 = r_j^2 \tag{6}$$

$$\begin{cases} x_{j\min} = x'_{oj} - r_j \\ x_{j\max} = x'_{oj} + r_j \\ y_{j\min} = y'_{oj} - r_j \\ y_{j\max} = y'_{oj} + r_j \end{cases} \tag{7}$$

where r_j represents the radius of the minimum covering circle of region j and $[x_{j\,\min}, x_{j\,\max}] \times [y_{j\,\min}, y_{j\,\max}]$ represents its minimum circumscribed rectangle. The construction result is shown in Figure 2b.

Algorithm 1 Data partition.

Input: Moving objects dataset X, number of regions K, maximum iterations M, convergence threshold E.

Output: K region information $[x_{j\,\min}, x_{j\,\max}] \times [y_{j\,\min}, y_{j\,\max}]$, centroids' information of K regions $O'_j(x'_{oj}, y'_{oj}, v'_{xoj}, v'_{yoj})$.

1: Randomly select K moving objects as initial clustering centroids O_j, $j = 1, 2, \ldots, K$.
2: Initialize the number of iterations t=0.
3: Initialize $\Delta = \infty$.
4: **while** $t < M$ and $\Delta \geq E$ **do**
5: $X_i \in C^{(i)}$
6: $O_j \longleftarrow O'_j$
7: Update Δ, $t = t + 1$
8: **end while**
9: Build the minimum covering circle and the minimum circumscribed rectangle for each region.
10: **return** $[x_{j\min}, x_{j\max}] \times [y_{j\min}, y_{j\max}]$, O'_j.

3.2. Dimensionality Reduction

The dimensionality reduction function D must satisfy the following conditions:

$$\sup D(X_i) < \inf D(X_j) \tag{8}$$

where $X_i \in C_i$, $X_j \in C_j$, and $i < j$. In the dimensionality-reduction process, regions must be initially sorted and numbered. This is achieved by calculating the Lebesgue measure [16] of the rectangle formed by the center of each region and the coordinate axes. These measures are then compared, and the regions are sorted in ascending order. Subsequently, each region is assigned a unique identifier N_i. To ensure the effectiveness of this approach, we must guarantee that every point within a region has a dimension reduction value that falls within the interval $[N, N+1)$. Consequently, the dimensionality reduction function must possess three essential properties:

- Continuity: The dimensionality reduction value should be close for moving objects that are proximate in space, ensuring a smooth transition in the reduced dimensions.
- Boundedness: The dimension reduction value for any moving object within a region must lie within the interval $[N, N+1)$.
- Consistency: For any rectangle, the dimension reduction values at the lower left and upper right corners should accurately represent the range of dimension reduction values for the moving objects within that rectangle.

Property 3 is established to simplify the future query process and ensure high recall, which will be demonstrated in the forthcoming Section 3.4.

It is logical to compare the Lebesgue measure of each point within the region to the Lebesgue measure of the rectangle defined by the left and bottom boundaries of the region. According to the dimensionality reduction approach described in Equation (9), Equation (8) and the three properties above can be satisfied.

$$D(X_i) = N_i + \frac{\mu(H_i)}{\mu(C_i)} \tag{9}$$

where μ is the Lebesgue measure on R^2, $X_i \in C_i = [x_{i\min}, x_{i\max}] \times [y_{i\min}, y_{i\max}]$, $H_i = [x_{i\min}, x_i] \times [y_{i\min}, y_i]$. However, there is an issue with points on the region's boundary: their Lebesgue measure is 0, rendering them unsupportive for future queries. Consequently, we introduce a new method for dimensionality reduction. This approach utilizes a straightforward linear dimensionality reduction, as detailed in Equation (10):

$$D(X_i) = N_i + \left(\beta \times \frac{x_i - x_{\text{imin}}}{x_{\text{imax}} - x_{\text{imin}}} + (1-\beta) \times \frac{y_i - y_{\text{imin}}}{y_{\text{imax}} - y_{\text{imin}}}\right) \tag{10}$$

where $0 \leq \beta \leq 1$. In this approach, the value of α determines the relative impact of the x and y directions on the dimensionality reduction value. Equation (10) not only fulfills the previously mentioned conditions and properties, but also resolves the issue that Equation (9) cannot compute the dimension reduction value for points on the boundary. Subsequently, the moving objects are sorted based on the dimensionality reduction values. With calculating in Equation (10), the dimensionality reduction values for all moving objects in each region is ensured to be within a specific segment. Consequently, the dimension reduction values of moving objects in different regions will not overlap.

3.3. Learning Piecewise Function

This subsection will elaborate on the process by which FurMoLi constructs the learned index. After the partition of data regions and the reduction of data dimensions as discussed in the previous subsections, the initial data of moving objects are organized into K adjacent and disjoint regions within a one-dimensional array, based on their position distribution and motion state. For each region, the dimensionality reduction value of each moving object, termed the key value, serves as the input, while the corresponding position of this key value in the array, known as the index value, is used as the training label. A linear regression model is then developed as the learned index for each region. Consequently, a piecewise linear learned index is trained for the entire dataset of moving objects.

Figure 3 illustrates an example of the Key-Index CDF obtained using the methods described in the previous two subsections for a moving object dataset. The relationship between the key value and the index appears complicated, as depicted in the figure. Utilizing the approach from the literature [28] would make it challenging to simulate this complexity with a neural network, particularly for large datasets. Instead, we employed a piecewise linear function to fit this relationship for several reasons:

- The parameter size of the piecewise linear function is significantly smaller than that of a neural network, which reduces the space needed for learning the index structure;
- Constraining the piecewise linear function to be monotone is simpler;
- The training time for piecewise linear functions is much shorter than for neural networks;
- During the future query phase, the piecewise linear function can expedite computations and enhance query efficiency.

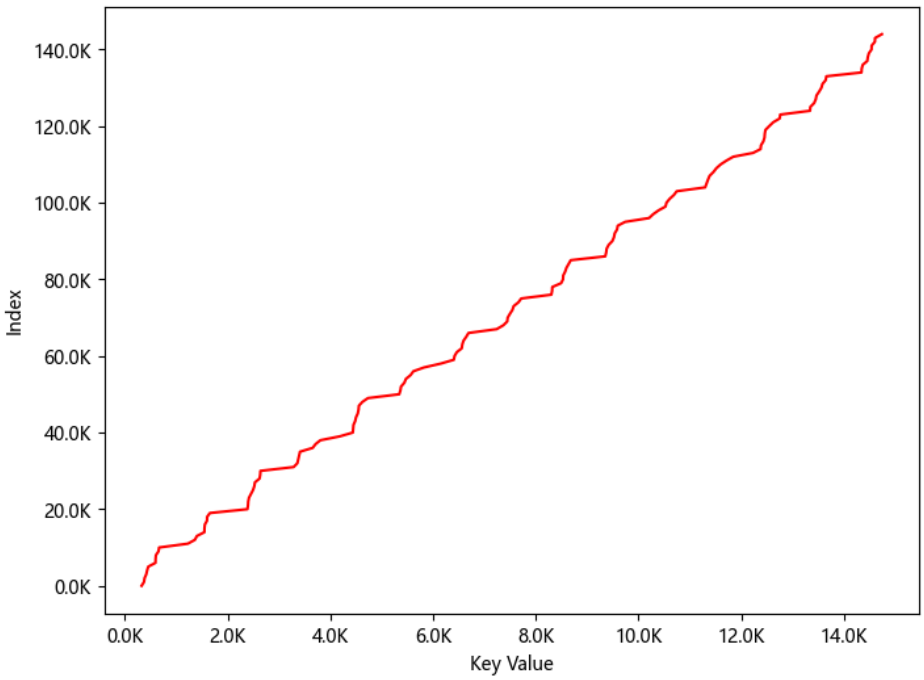


Figure 3. Example of Key-Index CDF.

For a region C_i, let $M_n = D(X_n)$, $X_n \in C_i$, and I_n be the corresponding index value of M_n, where n is the number of moving objects in C_i, $M_i \leq M_j$, $1 \leq i < j \leq n$. Let $\mathbf{M} = (M_1, M_2, \ldots, M_n)^T$ and $\mathbf{I} = (I_1, I_2, \ldots, I_n)^T$. Train a linear regression function $f_i(x)$ for region C_i with $\mathbf{M}$ as the model input and $\mathbf{I}$ as the model training label:

$$f_i(x) = a_i x + b_i \tag{11}$$

where a_i and b_i are the parameters of the model, $x \in \mathbf{M}$, $1 \leq i \leq K$. To ensure that Equation (11) is monotonically increasing, it is essential to guarantee that $a_i \geq 0$. We employed the least squares method to determine the parameters and constructed the loss function as Equation (12):

$$L_i = \sum_{j=1}^{n} \left(f_i(M_j) - I_j\right)^2 \tag{12}$$

To determine the corresponding parameters, it is essential to minimize the loss function. The gradient descent method is employed to calculate this minimization. By taking the partial derivative of Equation (12) and setting it to zero, the sum of squared differences between the predicted and actual values is minimized. The values of the parameters a_i and b_i for the linear regression model, which correspond to region C_i, are shown in Equation (13):

$$\begin{cases} \frac{\partial L_i}{\partial a_i} = 0 \\ \frac{\partial L_i}{\partial b_i} = 0 \end{cases} \Rightarrow \begin{cases} a_i = \frac{\sum_{j=1}^{n} M_j I_j - n\bar{M}\cdot\bar{I}}{\sum_{j=1}^{n} M_j^2 - n(\bar{M})^2} \\ b_i = \bar{I} - a_i \bar{M} \end{cases} \tag{13}$$

where $\bar{M}$ and $\bar{I}$ are the average values of $\mathbf{M}$ and $\mathbf{I}$.

Ultimately, the multi-segment linear model, as depicted in Figure 2c, is constructed. Since the prediction function invariably contains errors, similar to the RMI [9], a maximum error threshold ε_i must be established for each segment of the learned index, which is defined as the maximum difference between the predicted and actual index values.

$$\varepsilon_i = \max_j \left| f_i(M_j) - I_j \right| \tag{14}$$

Following the described construction process, a linear regression model is developed for the remaining $K-1$ regions, culminating in a global piecewise linear learned index. Finally, only the model parameters and error thresholds of the K-segment linear learned index need to be stored. In the next subsection, we will utilize $f_i(x)$ and the corresponding error threshold ε_i for future range query and future KNN query on moving objects.

3.4. Future Range Query

Since the learned index is constructed based on the current time, we adopted B^x-tree method to facilitate future range queries. For a future query window $Q[x_{q\,\min}, x_{q\,\max}] \times [y_{q\,\min}, y_{q\,\max}]$ at time t_q, we implemented future queries by expanding Q, as outlined in Algorithm 2. This approach ensures the recall rate of future queries and is more efficient than the MBR expansion method used by TPR-tree [3]. Detailed proofs are provided in the literature [17], and this paper offers only a brief explanation, as depicted in Figure 4. Q is a query window at time t_q in the future. The moving objects p_1, p_2 at t_0 move to p_1', p_2'at t_q, which are in the query window at t_q. In order to achieve this result, the B^x-tree assigns Q the maximum rates v_{rmax}, v_{lmax}, v_{umax}, and v_{dmax} of all moving objects in the four directions x^+, x^-, y^+, and y^-. Expand the query window to $Q'[x'_{q\,\min}, x_{q\,\max}]' \times [y'_{q\,\min}, y'_{q\,\max}]$ by Equation (15):

$$\begin{cases} x'_{q\min} = x_{q\min} - v_{l\max} \cdot (t_q - t_0) \\ x'_{q\max} = x_{q\max} + v_{r\max} \cdot (t_q - t_0) \\ y'_{q\min} = y_{q\min} - v_{d\max} \cdot (t_q - t_0) \\ y'_{q\max} = y_{q\max} + v_{\max} \cdot (t_q - t_0) \end{cases} \tag{15}$$

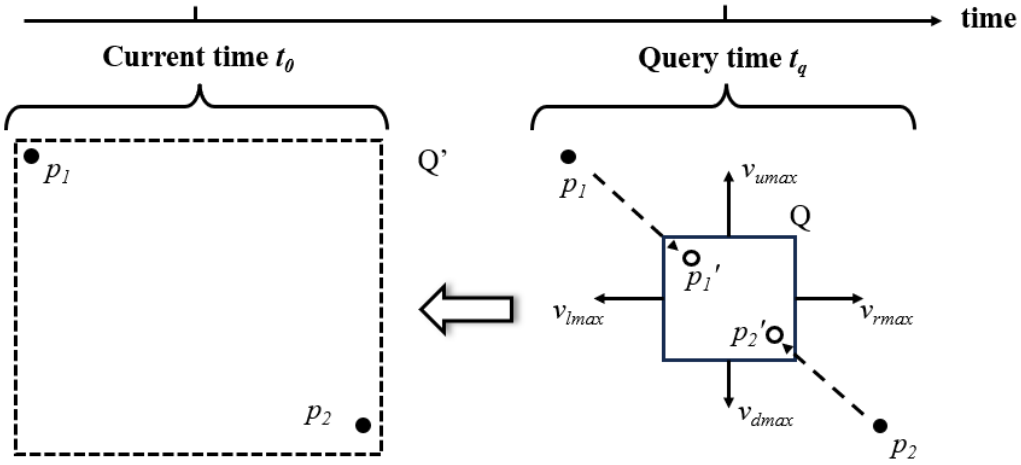


Figure 4. Illustration of query window expansion.

Because this query method is not constrained by the basic structure of the index, it is well suited to the learned index structure. Leveraging the consistency of the dimensionality reduction function introduced in Section 3.2, the issue of future range query for moving objects based on the learned index can be efficiently addressed. By determining the coordinates of the lower left L and upper right R corners of the intersection between the expanded query window Q' and the regions, computing the dimensionality reduction values, the moving objects within the query window Q at time t_q can be identified using the region's learned index $[f_i(D(L)) - \varepsilon_i, f_i(D(R)) + \varepsilon_i]$. In the following, we will prove the recall of the preceding query process, which is also the consistency of the dimensionality reduction function:

Let the intersection of the expanded query window Q' and the region C_i be as rectangle P, with the lower left corner coordinates $L(x_{pmin}, y_{pmin})$ and the upper right corner coordinates $R(x_{pmax}, y_{pmax})$. For $\forall A(x, y) \in P$:

$$\begin{cases} x_{p\min} \le x \le x_{p\max} \\ y_{p\min} \le y \le y_{p\max} \end{cases} \Rightarrow D(L) \le D(A) \le D(R) \tag{16}$$

Based on Equation (16), for any point inside P, its dimensionality reduction value must be between the dimensionality reduction values in the lower left corner and upper right corner of P.

Since simple linear dimensionality reduction does not account for the dense distribution of data points of moving objects within intersecting rectangles, a small number of spatially non-adjacent points become adjacent in the one-dimensional array. Additionally, due to the error threshold ε_i, a few moving objects that do not belong to the intersecting rectangle P will be returned in the range $[f_i(D(L)) - \varepsilon_i, \quad f_i(D(R)) + \varepsilon_i]$, as depicted in Figure 5. To address this, a straightforward filtering step is added to remove moving objects not within Q', as shown in Figure 2d. Given that the number of objects requiring filtering is significantly smaller than the initial dataset, the query efficiency of the algorithm will not be substantially affected.

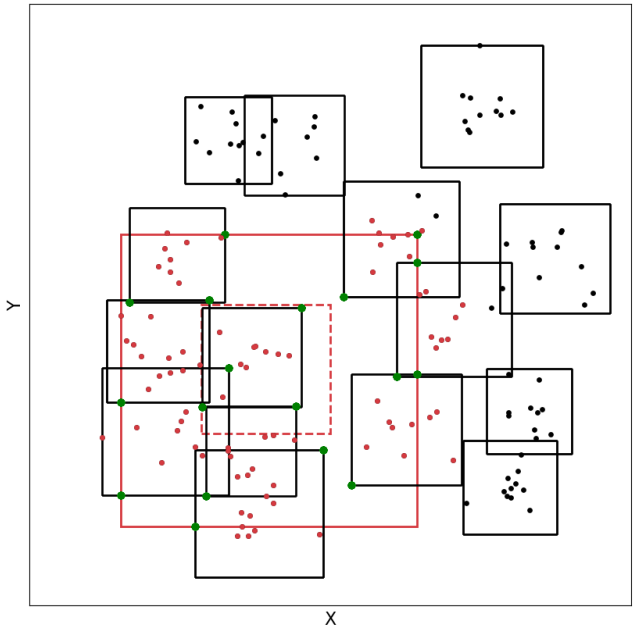


Figure 5. Example of querying additional objects. The points returned are red. The green points are the bottom left and top right corners of the intersecting rectangles. The dashed rectangle is the original query window, and the realization rectangle is the expanded query window.

Algorithm 2 Future range query.

Input: Query time t_q, query window $Q[x_{q\min}, x_{q\max}] \times [y_{q\min}, y_{q\max}]$.
Output: Moving objects $X_q \in Q$.
1: Get v_{rmax}, v_{lmax}, v_{umax}, v_{dmax}.
2: Expand $Q \longrightarrow Q'$.
3: $Q' \cap C_i = P_i$.
4: Get L_i, R_i of P_i.
5: Calculate $[f_i(D(L)) - \varepsilon_i, f_i(D(R)) + \varepsilon_i]$, and obtain the moving objects.
6: Filter moving objects $\notin Q'$
7: **return** X_q.

3.5. Future KNN Query

A future k-nearest neighbor (KNN) query involves, given a moving object $q(x_q, y_q)$, querying the k nearest moving objects to q at a future time t_q. On the basis of Section 3.4, this paper describes an iterative expansion of the future range query window until a sufficient number of k-nearest moving objects are returned. The specific process is outlined in Algorithm 3.

Initially, a positive query rectangle Q_{k1} is constructed with q and $2r_q$ as the side length, where the calculation for r_q is provided in Equations (17) and (18) [29].

$$r_q = \frac{D_k}{k} \tag{17}$$

$$D_k = \frac{2}{\sqrt{\pi}} \left[1 - \sqrt{1 - \left(\frac{k}{N} \right)^{1/2}} \right] \tag{18}$$

where D_k is the estimated distance between q and its k^{th} nearest neighbor moving object and N is the total number of moving objects. For the future time t_q, Q_{k1} is expanded to Q'_{k1} according to Algorithm 2. If Q'_{k1} contains at least k moving objects within the inscribed circle of Q_{k1} at t_q, the process stops. Otherwise, Q_{k1} is expanded to Q_{k2} by r_q, and Q_{k2}

is further expanded to Q'_{k2} following Algorithm 2. The search then proceeds in the area between $Q'_{k2} - Q'_{k1}$, iterating until the k nearest moving objects to q are identified.

Algorithm 3 Future KNN query.

Input: Query time t_q, moving object $q(x_q, y_q)$, number of nearest neighbors k.
Output: k-nearest moving objects to q.

```
Calculate r_q, and obtain Q_k1.
Expand Q_k1 ⟶ Q'_k1.
Initialize i = 1.
while 1 do
  if i = 1 then
    Search moving objects in Q'_k1.
  else
    Search moving objects in Q'_ki − Q'_ki−1.
  end if
  if k moving objects exist in inscribed circle of Q_k1 then
    Break.
  else
    i = i + 1.
    Expand Q_ki−1 ⟶ Q_ki.
    Expand Q_ki ⟶ Q'_k1
  end if
end while
return k-nearest moving objects.
```

4. Experimental Results

4.1. Experimental Settings and Datasets

In this section, we assess the performance of FurMoLi through a series of experiments, comparing it with traditional moving objects index structures such as TPR-tree and B^x-tree. Our evaluation focuses on the index structure construction efficiency, space overhead, and future query performance, validating the effectiveness of FurMoLi. Based on the experimental results presented in [17], the B^x-tree that utilizes the H-curve outperforms in all aspects. Therefore, in this experiment, the B^x-tree was constructed using the H-curve.

The experiments were carried out on a 12th Gen Intel(R) Core(TM) i7-12700KF 3.60 GHz desktop computer equipped with 32GB of memory and 1.5TB of disk space. To ensure the fairness of the experiments, all three models were implemented using Python 3.10 and tested exclusively on the CPU.

We employed the same data generator as previously used for simulating the moving object datasets, ensuring experimental fairness when evaluating TPR-tree and B^x-trees as per the methodologies described in [3,17]. This approach facilitates direct comparisons of our results with those studies. The generator is defined as follows [3]:

Consider a scenario wherein N moving objects traverse a vast spatial expanse measuring 1000 × 1000 km^2. ND destinations are uniformly positioned throughout this region, serving as vertices of an intricately connected graph with bidirectional thoroughfares. Throughout the simulation, the object count remains constant—neither diminishing nor augmenting. These objects are initially positioned at random junctures along any given roadway. Each object is impartially allocated to one of three max speed categories—0.75, 1.5, or 3 km/min (equivalent to 45, 90, and 180 km/h, respectively)—ensuring a uniform probability distribution across these velocities. The movement dynamics are segmented as follows: the initial one-sixth of the journey is dedicated to acceleration from standstill to peak velocity; the subsequent two-thirds of the route are traversed at this constant maximum speed; the concluding segment, constituting the final sixth, involves deceleration. Upon arrival at a designated destination, an object is promptly reassigned a new destination at random.

For each dataset, we initialized the index structure at time 0 and monitored the query efficiency after 10 time units (if the time is set too long, there is no predictive significance). The parameters applied are detailed in Table 3, with bold values indicating the default settings used.

Table 3. Parameter settings.

Parameter	Setting
N (number of moving objects)	**100 K** , 200 K, . . . , 800 K
K (number of regions)	10, **20**, 50, 100
Range query window size	**4**%, 10%, 25%, 40%, 60%, 80%
k (number of nearest neighbors)	**10**, 20, 40, 80, 100
ND (number of destinations)	10, **20**, 30, 40, 50

4.2. Storage Overhead

Storage overhead is crucial for index structures of moving objects. In the big data era, the storage overhead associated with traditional index structures often becomes excessively large, occasionally surpassing the space occupied by the underlying data themselves. Table 4 shows the space occupied by three different index structures across varying dataset sizes. It is clear that both B^x-tree and TPR-tree exhibit growth in space consumption as the dataset size increases. In contrast, the space required by FurMoLi remains relatively constant and is up to 4 or 5 orders of magnitude smaller than that of the other two index structures.

Table 4. Storage overhead for different index structures (KBytes).

N \ Index Structure	TPR-Tree	B^x-Tree	FurMoLi
100 K	6692.86	18,970.73	1.96
200 K	13,205.50	37,920.91	1.96
300 K	19,943.42	61,677.18	1.96
400 K	26,701.82	82,158.05	1.94
500 K	33,169.41	102,703.15	1.94
600 K	39,624.70	123,195.17	1.95
700 K	46,538.75	154,616.77	1.95
800 K	52,912.13	176,723.72	1.95

This difference arises because traditional index structures store the entire data structure, including nodes and pointers, whereas FurMoLi requires only the storage of parameters a, b, and ε from the segmented learning model. With the number of regions K set to 20, only 60 parameter values need to be stored, significantly reducing the storage overhead. The storage overhead for FurMoLi remains relatively constant, as it is solely dependent on the value of regions K and the number of parameters, rather than the volume of the underlying data. This relationship is demonstrated in Table 5.

Table 5. Impact of different K with different dataset size on FurMoLi storage overhead (KBytes).

K \ *N*	100 K	200 K	300 K	400 K	500 K
10	0.973	0.973	0.967	0.976	0.980
20	1.959	1.959	1.949	1.944	1.943
50	4.883	4.874	4.891	4.906	4.903
100	9.814	9.775	9.770	9.779	9.790

4.3. Building Efficiency

Figure 6 illustrates the index construction time for three index structures across different dataset sizes. As the dataset size increases, the construction time for all three structures also increases. Among them, the TPR-tree exhibits the largest increase in construction time. In contrast, both B^x-tree and FurMoLi show relatively smaller increases. Specifically, the construction time for FurMoLi is slightly higher than that of B^x-tree. This is because FurMoLi's construction involves two stages of machine learning—clustering and regression. Nevertheless, FurMoLi remains relatively efficient on index construction time.

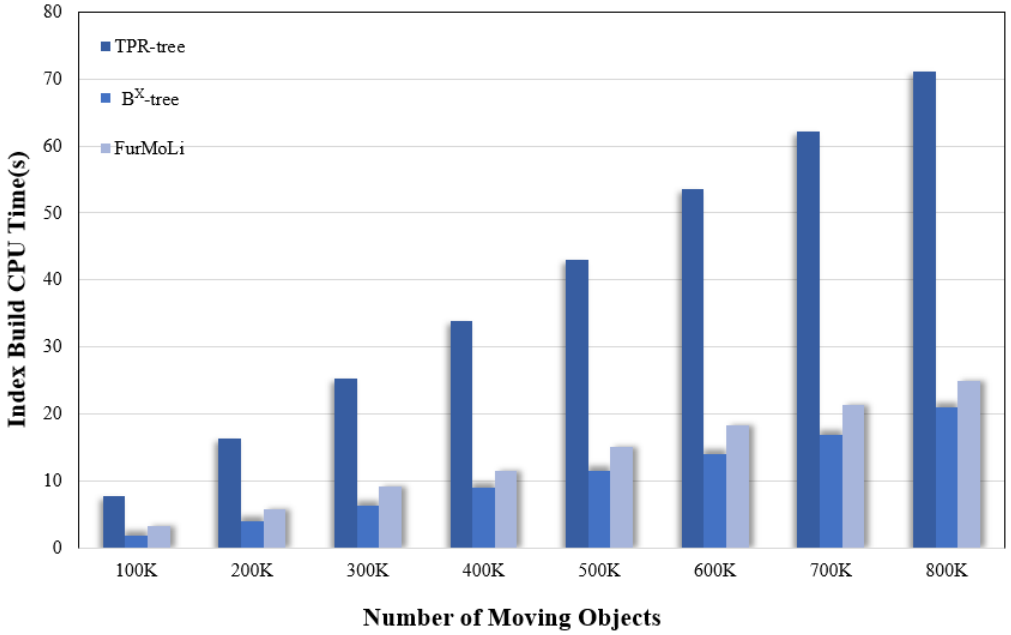


Figure 6. Performance comparison of building efficiency.

4.4. Future Range Query

Figure 7 illustrates the efficiency of the three index structures for the processing future range queries across various dataset sizes. As the data volume grows, the query time for all three structures also increases due to the increase of the objects inside the query window. However, TPR-tree and B^x-tree exhibit significant increases, while FurMoLi shows a more gradual rise. This demonstrates that FurMoLi can scale more effectively while maintaining consistent performance compared to the two traditional index structures. Meanwhile, FurMoLi is the most efficient, consuming only 41.6% of TPR-tree and 34.7% of B^x-tree in query time on average. This superior performance is attributed to FurMoLi's ability to learn the data distribution better and train a suitable model using a piecewise-learned index. Additionally, unlike traditional index structures that rely on traversal queries, FurMoLi primarily performs calculations, resulting in lower time complexity.

Next, we discuss how the size of the query window affects query efficiency. We analyzed the query performance of three index structures with a dataset of size 100K by adjusting the ratio of the future query window area to the total space area. Figure 8 shows that query time increases with the query window size. For TPR-trees and B^x-trees, larger query windows contain more moving objects, requiring more nodes to be accessed. For FurMoLi, larger query windows lead to more intersecting regions, an increased amount of filtered data, and more computation. According to the comparison results, FurMoLi consistently exhibits the best query efficiency, regardless of the query window size.

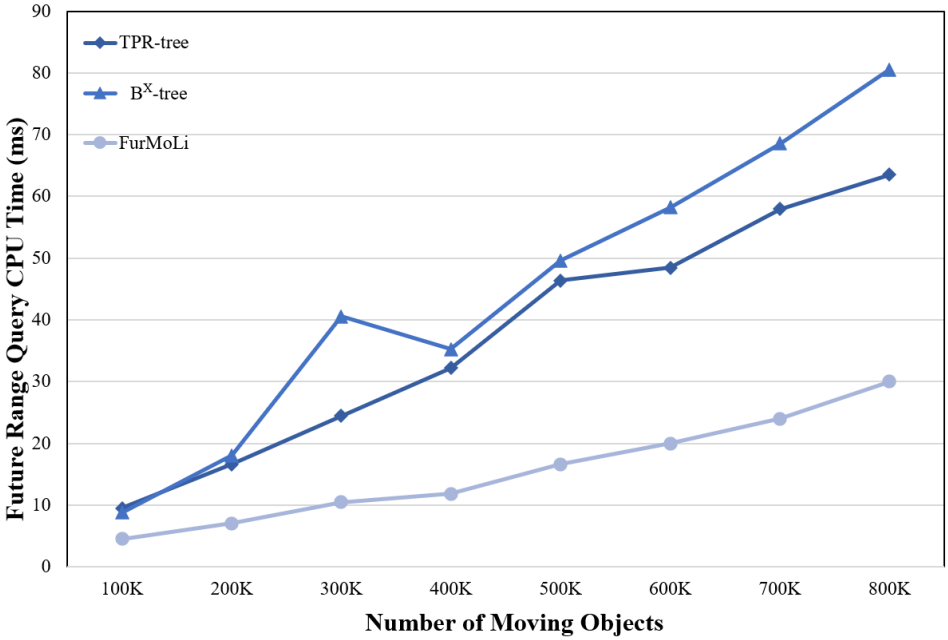


Figure 7. Performance comparison of future range query.

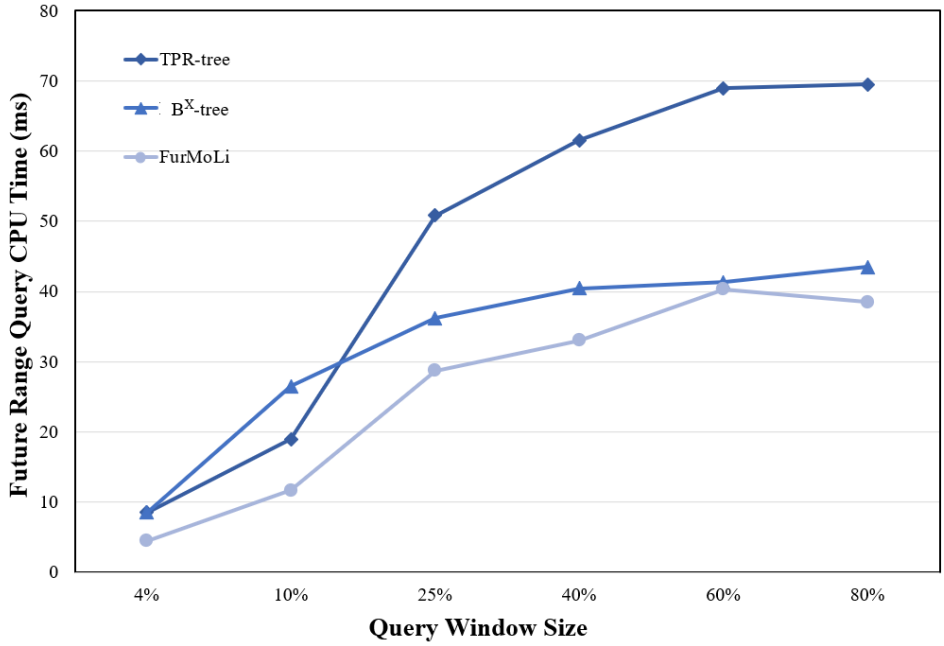


Figure 8. Effect of query window sizes on future range query.

We also investigated the impact of the data distribution on the query efficiency of the three index structures by varying the number of destinations (ND) during data construction. Figure 9 shows that the query cost of all three index structures slowly increases as the number of destinations increases, with only slight differences in milliseconds. FurMoLi consistently demonstrates the best query efficiency. However, a notable increase in query cost occurs at $ND = 50$. This is because, as ND increases, the data become more disorganized, worsening the clustering effect and reducing query efficiency.

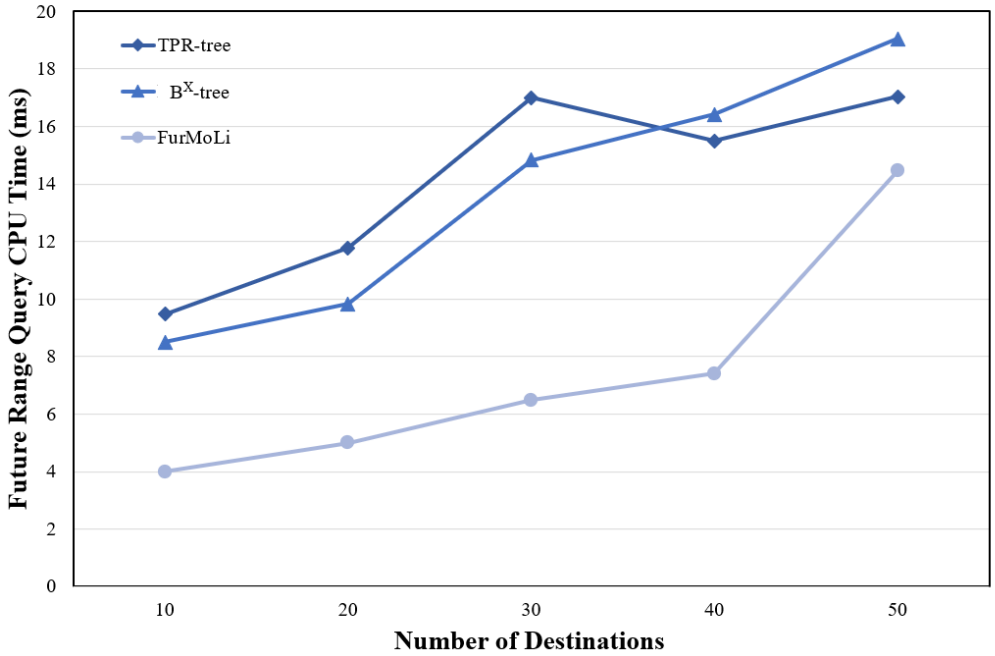


Figure 9. Effect of data distribution on future range query.

4.5. Future KNN Query

For the future KNN query, the efficiency was evaluated similarly to the future range query. Figures 10 and 11 show the effects of the dataset size and data distribution on the future KNN query performance. Since FurMoLi's future KNN query is based on the future range query, the results are similar to those exhibited by the future range query.

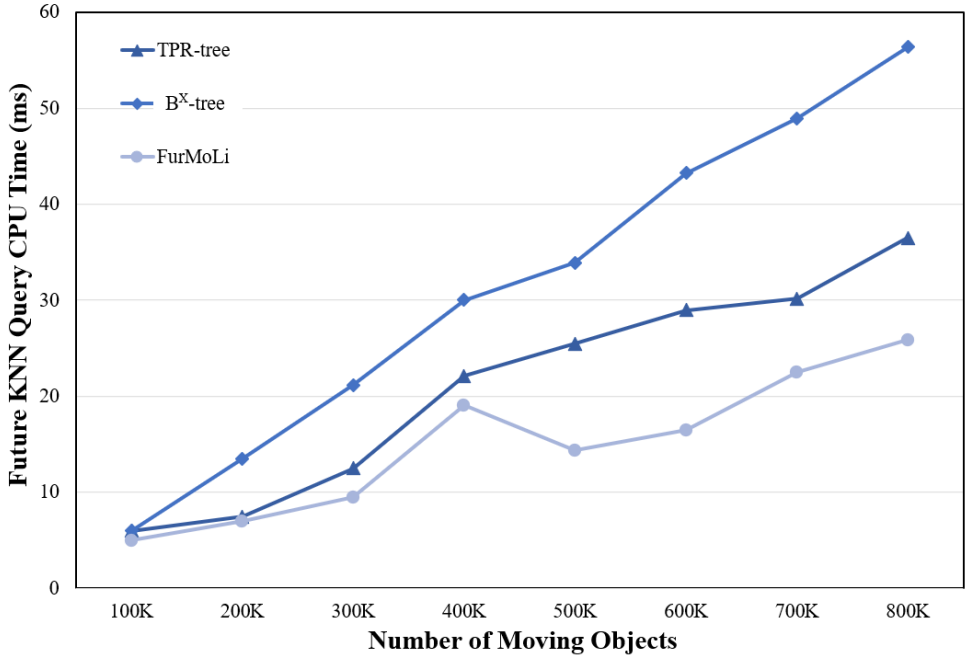


Figure 10. Performance comparison of future KNN query.

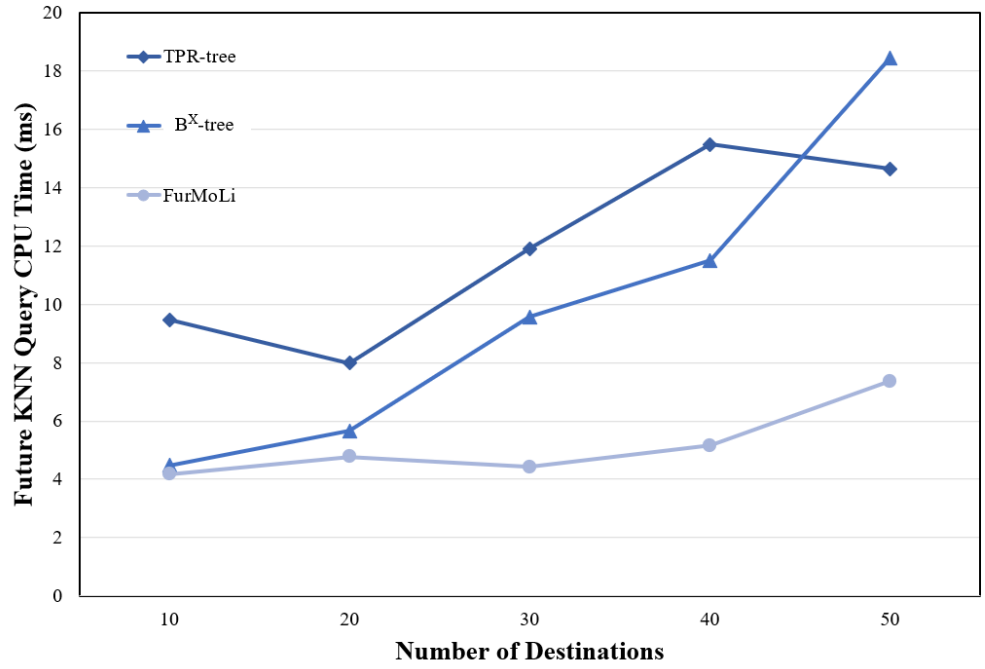


Figure 11. Effect of data distribution on future KNN query.

Figure 12 illustrates the future KNN query efficiency of the three index structures for different k values. As k increases, the query time for TPR-tree increases the most, while B^x-tree and FurMoLi show slower increases. FurMoLi remains the most efficient across different k values.

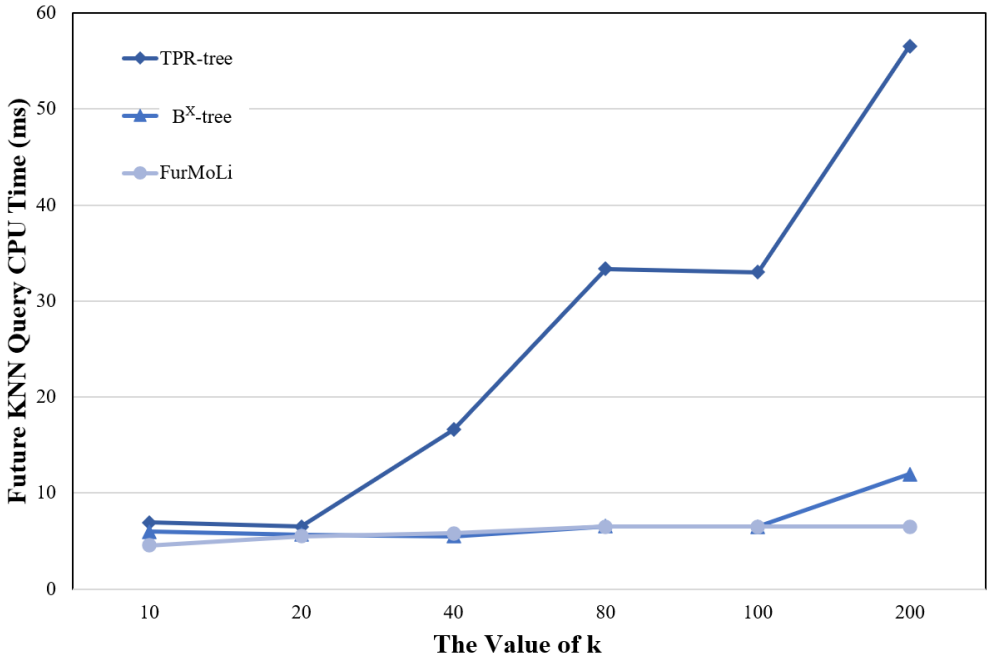


Figure 12. Effect of k on future KNN query.

5. Conclusions

In this paper, we propose FurMoLi—a novel future query technique for moving objects based on the learned index. To the best of our knowledge, FurMoLi is the first method that uses the learned index for the future query of moving objects. The experimental results

demonstrated that FurMoLi outperforms the traditional moving object indexes, TPR-tree and B^x-tree, particularly in terms of query time for the future range query and future KNN query. Notably, FurMoLi also offers significant advantages in storage overhead. Additionally, FurMoLi exhibits excellent scalability, maintaining consistent query and storage performance even as the data volume increases significantly. Finally, FurMoLi demonstrates strong adaptability, maintaining good query performance even when the number of destination (data distribution) changes.

In this work, we did not consider the update mechanism of FurMoLi, which provides a direction for subsequent research. When the data change significantly, a mature update mechanism can avoid the negative impact of index rebuilding. Meanwhile, FurMoLi utilizes a simple linear regression model. In future work, more advanced machine learning and deep learning models could be explored to build the learned index, thereby enhancing both query efficiency and index construction. At the same time, the dimensionality reduction method used in this paper ensures high recall, but achieving precision requires an additional filtering step, which reduces query efficiency. Future work will aim to explore algorithms that can guarantee both recall and precision. Ultimately, this paper only addresses two basic queries: future range query and future KNN query. Exploring the use of FurMoLi for the future continuous query and future closest pairs query is another important direction for future research.

Author Contributions: Conceptualization, J.Y., C.Z. and W.T.; methodology, J.Y. and C.Z.; software, J.Y.; validation, B.G. and H.H.; writing—original draft preparation, J.Y.; writing—review and editing, J.Y., C.Z., W.T., B.G. and S.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Theiss, A.; Yen, D.C.; Ku, C. Global Positioning Systems: An analysis of applications, current development and future implementations. *Comput. Stand.* **2005**, *27*, 89–100. [CrossRef]
2. Goodchild, M.F. Twenty years of progress: GIScience in 2010. *J. Spat. Inf. Sci.* **2010**, *1*, 3–20. [CrossRef]
3. Saltenis, S.; Jensen, C.S.; Leutenegger, S.T.; López, M.A. Indexing the Positions of Continuously Moving Objects. In Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, TX, USA, 16–18 May 2000; Chen, W., Naughton, J.F., Bernstein, P.A., Eds.; ACM: New York, NY, USA, 2000; pp. 331–342. [CrossRef]
4. Tao, Y.; Papadias, D.; Sun, J. The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries. In Proceedings of the 29th International Conference on Very Large Data Bases, VLDB 2003, Berlin, Germany, 9–12 September 2003; Freytag, J.C., Lockemann, P.C., Abiteboul, S., Carey, M.J., Selinger, P.G., Heuer, A., Eds.; Morgan Kaufmann: Burlington, MA, USA, 2003; pp. 790–801. [CrossRef]
5. Guttman, A. R-Trees: A Dynamic Index Structure for Spatial Searching. In Proceedings of the SIGMOD'84, Boston, MA, USA, 18–21 June 1984; Yormark, B., Ed.; ACM Digital Library: New York, NY, USA, 1984; pp. 47–57. [CrossRef]
6. Beckmann, N.; Kriegel, H.; Schneider, R.; Seeger, B. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, USA, 23–25 May 1990; Garcia-Molina, H., Jagadish, H.V., Eds.; ACM Press: New York, NY, USA, 1990; pp. 322–331. [CrossRef]
7. Zhang, H.; Andersen, D.G.; Pavlo, A.; Kaminsky, M.; Ma, L.; Shen, R. Reducing the Storage Overhead of Main-Memory OLTP Databases with Hybrid Indexes. In Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, 26 June–1 July 2016; Özcan, F., Koutrika, G., Madden, S., Eds.; ACM: New York, NY, USA, 2016; pp. 1567–1581. [CrossRef]
8. Wu, X.; Ni, F.; Jiang, S. Wormhole: A Fast Ordered Index for In-memory Data Management. In Proceedings of the Fourteenth EuroSys Conference 2019, Dresden, Germany, 25–28 March 2019; Candea, G., van Renesse, R., Fetzer, C., Eds.; ACM: New York, NY, USA, 2019; pp. 18:1–18:16. [CrossRef]
9. Kraska, T.; Beutel, A.; Chi, E.H.; Dean, J.; Polyzotis, N. The Case for Learned Index Structures. In Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, 10–15 June 2018; Das, G., Jermaine, C.M., Bernstein, P.A., Eds.; ACM: New York, NY, USA, 2018; pp. 489–504. [CrossRef]

10. Li, P.; Lu, H.; Zheng, Q.; Yang, L.; Pan, G. LISA: A Learned Index Structure for Spatial Data. In Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, Online Conference, Portland, OR, USA, 14–19 June 2020; Maier, D., Pottinger, R., Doan, A., Tan, W., Alawini, A., Ngo, H.Q., Eds.; ACM: New York, NY, USA, 2020; pp. 2119–2133. [CrossRef]
11. Nathan, V.; Ding, J.; Alizadeh, M.; Kraska, T. Learning Multi-Dimensional Indexes. In Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, Online Conference, Portland, OR, USA, 14–19 June 2020; Maier, D., Pottinger, R., Doan, A., Tan, W., Alawini, A., Ngo, H.Q., Eds.; ACM: New York, NY, USA, 2020; pp. 985–1000. [CrossRef]
12. Qi, J.; Liu, G.; Jensen, C.S.; Kulik, L. Effectively Learning Spatial Indices. *Proc. VLDB Endow.* **2020**, *13*, 2341–2354. [CrossRef]
13. Wang, H.; Fu, X.; Xu, J.; Lu, H. Learned Index for Spatial Queries. In Proceedings of the 20th IEEE International Conference on Mobile Data Management, MDM 2019, Hong Kong, China, 10–13 June 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 569–574. [CrossRef]
14. Cai, P.; Zhang, S.; Liu, P.; Sun, L.; Li, C.; Chen, H. An Overview of Learned Index Technologies for Intelligent Database. *Chin. J. Comput.* **2023**, *46*, 51–69.
15. Li, Y.; Han, J.; Yang, J. Clustering moving objects. In Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, WA, USA, 22–25 August 2004; Kim, W., Kohavi, R., Gehrke, J., DuMouchel, W., Eds.; ACM: New York, NY, USA, 2004; pp. 617–622. [CrossRef]
16. Royden, H.L.; Fitzpatrick, P. Real Analysis. Featured Titles for Real Analysis Series. *Featur. Titles Real Anal. Ser.* **2010**, *32*.
17. Jensen, C.S.; Lin, D.; Ooi, B.C. Query and Update Efficient B+-Tree Based Indexing of Moving Objects. In Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004, Toronto, ON, Canada, 31 August–3 September 2004; Nascimento, M.A., Özsu, M.T., Kossmann, D., Miller, R.J., Blakeley, J.A., Schiefer, K.B., Eds.; Morgan Kaufmann: Burlington, MA, USA, 2004; pp. 768–779. [CrossRef]
18. Benetis, R.; Jensen, C.S.; Karciauskas, G.; Saltenis, S. Nearest and reverse nearest neighbor queries for moving objects. *VLDB J.* **2006**, *15*, 229–249. [CrossRef]
19. Pfoser, D.; Jensen, C.S.; Theodoridis, Y. Novel Approaches to the Indexing of Moving Object Trajectories. In Proceedings of the VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, Cairo, Egypt, 10–14 September 2000; Abbadi, A.E., Brodie, M.L., Chakravarthy, S., Dayal, U., Kamel, N., Schlageter, G., Whang, K., Eds.; Morgan Kaufmann: Burlington, MA, USA, 2000; pp. 395–406.
20. Procopiuc, C.M.; Agarwal, P.K.; Har-Peled, S. STAR-Tree: An Efficient Self-Adjusting Index for Moving Objects. In Proceedings of the Algorithm Engineering and Experiments, 4th International Workshop, ALENEX 2002, San Francisco, CA, USA, 4–5 January 2002; Revised Papers; Lecture Notes in Computer Science; Mount, D.M., Stein, C., Eds.; Springer: Berlin/Heidelberg, Germany, 2002; Volume 2409, pp. 178–193. [CrossRef]
21. Tao, Y.; Faloutsos, C.; Papadias, D.; Liu, B. Prediction and Indexing of Moving Objects with Unknown Motion Patterns. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, 13–18 June 2004; Weikum, G., König, A.C., Deßloch, S., Eds.; ACM: New York, NY, USA, 2004; pp. 611–622. [CrossRef]
22. Yiu, M.L.; Tao, Y.; Mamoulis, N. The B^{dual}-Tree: Indexing moving objects by space filling curves in the dual space. *VLDB J.* **2008**, *17*, 379–400. [CrossRef]
23. Kraska, T.; Alizadeh, M.; Beutel, A.; Chi, E.H.; Kristo, A.; Leclerc, G.; Madden, S.; Mao, H.; Nathan, V. SageDB: A Learned Database System. In Proceedings of the 9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, 13–16 January 2019; Online Proceedings. 2019. [CrossRef]
24. Davitkova, A.; Milchevski, E.; Michel, S. The ML-Index: A Multidimensional, Learned Index for Point, Range, and Nearest-Neighbor Queries. In Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, 30 March–2 April 2020; Bonifati, A., Zhou, Y., Salles, M.A.V., Böhm, A., Olteanu, D., Fletcher, G.H.L., Khan, A., Yang, B., Eds.; Open Proceedings: Konstanz, Germany , 2020; pp. 407–410. Available online: https://OpenProceedings.org (accessed on 31 May 2020)CrossRef]
25. Jagadish, H.V.; Ooi, B.C.; Tan, K.; Yu, C.; Zhang, R. iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.* **2005**, *30*, 364–397. [CrossRef]
26. Zhang, S.; Ray, S.; Lu, R.; Zheng, Y. SPRIG: A Learned Spatial Index for Range and kNN Queries. In Proceedings of the 17th International Symposium on Spatial and Temporal Databases, SSTD 2021, Virtual Event , 23–25 August 2021; Hoel, E., Oliver, D., Wong, R.C., Eldawy, A., Eds.; ACM: New York, NY, USA, 2021; pp. 96–105. [CrossRef]
27. Selim, S.Z.; Ismail, M.A. K-Means-Type Algorithms: A Generalized Convergence Theorem and Characterization of Local Optimality. *IEEE Trans. Pattern Anal. Mach. Intell.* **1984**, *6*, 81–87. [CrossRef] [PubMed]
28. Amato, D.; Bosco, G.L.; Giancarlo, R. On the Suitability of Neural Networks as Building Blocks for The Design of Efficient Learned Indexes. *arXiv* **2022**, arXiv:2203.14777.
29. Tao, Y.; Zhang, J.; Papadias, D.; Mamoulis, N. An Efficient Cost Model for Optimization of Nearest Neighbor Search in Low and Medium Dimensional Spaces. *IEEE Trans. Knowl. Data Eng.* **2004**, *16*, 1169–1184. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.