PPQ-Trajectory: Spatio-temporal Quantization for Querying in Large Trajectory Repositories

Shuang Wang  
University of Warwick  
Coventry, United Kingdom  
Shuang.Wang.1@warwick.ac.uk

Hakan Ferhatosmanoglu  
University of Warwick  
Coventry, United Kingdom  
Hakan.F@warwick.ac.uk

ABSTRACT

We present PPQ-trajectory, a spatio-temporal quantization based solution for querying large dynamic trajectory data. PPQ-trajectory includes a partition-wise predictive quantizer (PPQ) that generates an error-bounded codebook with autocorrelation and spatial proximity-based partitions. The codebook is indexed to run approximate and exact spatio-temporal queries over compressed trajectories. PPQ-trajectory includes a coordinate quadtree coding for the codebook with support for exact queries. An incremental temporal partition-based index is utilised to avoid full reconstruction of trajectories during queries. An extensive set of experimental results for spatio-temporal queries on real trajectory datasets is presented. PPQ-trajectory shows significant improvements over the alternatives with respect to several performance measures, including the accuracy of results when the summary is used directly to provide approximate query results, the spatial deviation with which spatio-temporal path queries can be answered when the summary is used as an index, and the time taken to construct the summary. Superior results on the quality of the summary and the compression ratio are also demonstrated.

PVLDB Reference Format:
Shuang Wang and Hakan Ferhatosmanoglu. PPQ-Trajectory: Spatio-temporal Quantization for Querying in Large Trajectory Repositories. PVLDB, 14(2): 215-227, 2021.
doi:10.14778/3425879.3425891

1 INTRODUCTION

With the prevalence of positioning devices and mobile services, massive amounts of location sequences are being generated continuously. Maintaining and querying small-sized representations of raw trajectory data are needed for a wide variety of applications, such as real-time traffic management [32] and intelligent transport systems [4].

Existing trajectory compression methods do not address this need for a number of reasons. First, many of them are defined for edge sequences in a road network [14, 18, 21, 38]. They require pre-processing steps of mapping raw GPS data to the road network structure, followed by transforming the map-matched location data to edge-based sequences. The mapping and transformation processes reduce accuracy and result in limited support for detailed queries. Second, most solutions perform offline compression over full trajectory data, with execution times usually undesirable for online applications. There is a need for scalable online compression. Third, the existing compressed representations can not be directly used to answer spatio-temporal queries without a costly decompression process.

To address these challenges, we present PPQ-trajectory, a spatio-temporal quantization-based solution to generate a compact representation and support a wide range of queries over large trajectory data. An overview of PPQ-trajectory is presented in Figure 1.

The first part of PPQ-trajectory is the partition-wise predictive quantizer (PPQ) that generates an error-bounded summary, consisting of the codebook and prediction coefficients for spatial and autocorrelation-based partitions. The second part is the coordinate quadtree coding (CQC) for the error space caused by the quantization, which enables an accurate reconstruction of the trajectories. These two parts form the summary for the trajectory data, as illustrated in Figure 1. The third part is the temporal partition-based index (TPI) for organizing the quantized spatio-temporal data. Given a query, TPI is used to prune the data space and generate a candidate list of trajectories, whose reconstructed points can be computed from the summary. Overall, PPQ-trajectory generates and uses an indexed summary over raw data sequences to support efficient analysis, ranging from simple queries, such as vehicles passing by a location \((x, y)\) at a given time \(t\), to more complex analytic tasks, such as predicting future positions of entities.

We evaluate PPQ-trajectory with respect to a number of performance measures: the quality of approximate query results, efficiency of exact queries, index building times, and compression ratio. We implemented several baselines, including the widely-used product quantization [19], residual quantization [8], REST [44], which is a recent reference based trajectory compression method, and TrajStore [10], an adaptive storage solution for trajectories.

![Figure 1: An Overview of PPQ-trajectory](image-url)
This paper makes the following contributions: (1) A spatio-temporal predictive quantizer, PPQ, is designed with an error-bounded codebook for each of the partitions, which are incrementally generated based on spatial and autocorrelation similarity. (2) Utilizing a quadtree structure and a padding process, CQC is developed to encode the relative positions of trajectory points with reconstructed ones for an accurate trajectory reconstruction. A local search strategy is presented to identify exact query results. (3) The temporal partition-based index dynamically reuses parts of the past index to support efficient spatio-temporal queries. (4) PPQ-trajectory answers spatio-temporal queries over raw data, without a full reconstruction of trajectories or accessing all of the candidate trajectories. (5) Experimental results demonstrate significant improvements achieved by PPQ-trajectory. For example, for approximate spatio-temporal queries, it is 3%–52% more accurate, compared to product quantization, residual quantization, and TrajStore. The mean absolute error (MAE) of PPQ-trajectory is a few or tens of meters, while the alternative approaches’ MAE values are orders of magnitude larger for the same size codebook. Significant improvements are also observed for the efficiency, index building times, and compression ratios.

The rest of the paper is organized as follows. The related work is in Section 2. Section 3 presents PPQ. In Section 4, we present CQC and the associated local search strategy. In Section 5, the temporal organization for the quantized data is presented. The effectiveness of PPQ-trajectory is verified via an extensive set of experiments in Section 6. Conclusion is provided in Section 7.

2 RELATED WORK

With the widespread adoption of location-based services, compressing trajectory data has become a prevalent area with high practical relevance [2, 21, 26]. While traditional compression methods aim to reduce the reconstruction error and improve compression ratio, the data management challenge is to design the compression method with the objective of answering queries efficiently, and supporting online querying directly over compressed data.

Road network-constrained trajectory compression has gained significant attention [14, 18, 21, 25, 36, 38]. The common approach is to map raw trajectories to road networks and compress the map-matched trajectories [20]. There is also significant attention on raw trajectory compression. SQUISH and SQUISH-E use a priority queue to remove redundant points [30, 31]. A bounded quadrant system (BQS) is developed in [25], which uses convex hull bounding to achieve trajectory compression. Based on BQS, [26] achieves streaming trajectory compression, and aging history data without overwriting. Another recent method transforms trajectories into vehicle state vector functions and generates an inverted index based on the road segments [4]. This approach requires road segment information and matching each road id with corresponding objects.

The solutions that we included in our experiments are TrajStore [10] and REST [44]. TrajStore aims compression via an adaptive spatial index and clustering the sub-trajectories. It recursively updates the index by merging, splitting or appending. REST is a recent compression-based method which compares trajectories with the sub-trajectories of a reference set. Generating a representative set is challenging especially under changing conditions, where it can fail to represent data from regions that lack enough samples.

Early work in this area applies traditional index structures for trajectory data. For example, STRIPES uses quadtrees to index the predicted positions of moving objects [35]. In [3], an index for trajectories is developed by indexing the coefficients of Chebyshev polynomials that represent trajectories. Most those methods focus on similarity queries and do not address efficient spatio-temporal database queries. Zheng et al., [45] index the reference-based trajectories with IR-tree [24], which is based on R-tree referring to the inverted files for sub-trajectories.

Quantization is a popular method for traditional compression, and for nearest neighbor searches on multi dimensional data, especially for multimedia and computer vision applications [12, 27, 33, 40, 43]. Predictive quantization has been applied for online summarization of multiple one-dimensional data streams [1]. The correlation among consecutive points is employed to predict current points, then the prediction errors are summarized into a smaller number of bits [1]. Product Quantization and Residual Quantization [8, 15, 19] have made significant impact on approximate nearest neighbor searching in computer vision applications. We included these two methods in our performance evaluation. There have been some work to use quantization for encoding trajectories [7], transforming differential trajectory points into strings for compression [29], and retaining information for trajectory prediction [5]. These methods adopt compression but with no particular support for efficient querying over compressed trajectories. Our goal is to quantize dynamic trajectories into an error-bounded and query friendly representation, where there is neither need to fully reconstruct nor traverse the full trajectories. Trajectory data is summarized online, exploiting their large-scale nature, for the purpose of efficient query processing.

3 ONLINE QUANTIZATION IN PPQ-TRAJECTORY

In this section, we present our spatio-temporal quantization based summarization process. The performance measures are the accuracy of results when the summary is used directly to provide approximate query results, the spatial deviation with which spatio-temporal path queries can be answered when the summary is used as an index, and the time taken to construct the summary. The quality of the summary and the compression ratio are also related measures. Table 1 summarizes the notation used throughout the paper. Basic definitions of trajectories and codebooks are as follows.

Definition 3.1. (Trajectory) A trajectory $T$ is a finite sequence of time-stamped positions in the form of $((x_1, y_1, t_1), (x_2, y_2, t_2), ..., (x_n, y_n, t_n))$, where $0 \leq t_i \leq t_j \leq t_n$ with $0 \leq i \leq j \leq n$.

Definition 3.2. (Error-bounded Codebook) Consider a codebook $C = \{C_1, ..., C_n\}$ where the set of trajectory points $T^i$ is indexed by codeword $C_i$. For any trajectory point $T^i_j \in T^i$, if $\|T^i_j - C_i\|_2 \leq \epsilon_1$, then the codebook $C$ is bounded with $\epsilon_1$.

3.1 Error-bounded Predictive Quantization

We first present the error bounded predictive quantizer (E-PQ) to obtain a compact codebook for trajectories. Predictive quantization (PQ) has been successfully applied for one-dimensional data streams.
where $\hat{e}_i$ is defined as:

$$\hat{e}_i = T_i - \tilde{T}_i, \quad \tilde{T}_i = \sum_{j=1}^{k} P_j |t| \hat{T}_{i-k}^j$$

The prediction errors $\{e_i^t\}$ can be summarized into an errorbounded codebook $C$ as:

$$\min \{|C| \quad s.t. \|e_i^t - C(b_i^t)\|_2 \leq \epsilon_1, b_i^t \in \{1, ..., V\}$$

A prediction function $f$ is learned over training data, and a codebook is generated via a vector quantizer [40] on the prediction errors by assigning them to the nearest centroids of their clusters. The range of the error $e_i^t = x_i^t - \tilde{x}_i^t$ is narrower than the original data which enables the errors to be quantized more effectively than the original data [1].

To estimate the trajectory points using their correlations, we define a prediction function as an extension to the case for one-dimensional streams [1]. For ease of demonstration, we define $f$ as a linear model that predicts $T_i^t$ using the previous $k$ samples. The prediction is computed as:

$$\min_{f} \sum_{t=1}^{N} \|T_i^t - f(T_i^{t-k})\|_2$$

where $T_i^t$ represents the position $(x_i^t, y_i^t)$ of $T_i$ at time $t$, $T_i^{t-k}$ is the sequence of the trajectory $T_i$ at time interval $[t-k, t-1]$, and $f$ denotes the prediction model.

The prediction error $\hat{e}_i^t$ is defined as:

$$\hat{e}_i^t = T_i^t - \tilde{T}_i^t, \quad \tilde{T}_i^t = \sum_{j=1}^{k} P_j |t| \hat{T}_{i-k}^j$$

The procedure of quantizing dynamic trajectories is summarized in Algorithm 1. In Line 3, the prediction coefficient $P_j |t|$ can be solved in a standard manner [1, 16]. Line 4 denotes the prediction of the $t$-th trajectory point by its previous $k$ reconstructed points. For the time $t \leq k$, $P_j |t|$ is set to zero. At Line 6, Incremental Quantizer represents the quantization process of Equation 3. E-PQ maps the trajectory data into $\{P_j |t\}$, $C$, $\{b_i^t\}$.

### 3.2 Partition-wise Predictive Quantization

We now present our quantizer that partitions trajectory points and applies E-PQ for each partition. The partition-wise predictive quantization (PPQ) is formulated as:

$$N_i^t = \{N_i^t_1, ..., N_i^t_q\}$$

$$\min \sum_{i=1}^{N_i^t} \|T_i^t - P_j |t| N_i^t_{j-k}^t\|_2, N_i^t_j \in N_i^t$$

In Equation 5, $N$ trajectory points $\{T_i^t\}$ are partitioned into $q$ subsets, where $N_i^t_j$ denotes the set of trajectory IDs assigned to the $j$-th partition. $P_j |t|$ is the prediction function for $N_i^t_j$. This enables the use of a single prediction function $f_j$ for trajectory points of $N_i^t_j$, and is resolved by Equation 6. Via $\{f_1, ..., f_q\}$, the correlation among consecutive trajectory points in every $N_i^t_j \in N_i^t$ is modeled by a specific $f_j$, then the dynamic range of prediction errors is further narrowed down. When $q = 1$, Equation 1 and Equation 6 become the same. Similarly for the $\{e_i^t\}$ obtained from multiple predictions, they are summarized with Equation 3.

#### 3.2.1 Partitioning for Grouped Modeling

We partition the trajectory points using their spatial and autocorrelation similarities. Tobler’s first law of geography indicates “everything is related to everything else, but nearby things are more related than distant things” [39]. Hence, assigning trajectory points based on spatial proximity is a natural approach to be able to use the same $f_j$ to

### Table 1: Summary of Notation

| Variable | Definition |
|----------|------------|
| $T_i$ | the $i$-th trajectory |
| $T_i^t$ | trajectory point of $T_i$ at time $t$ |
| $\tilde{T}_i^t$ | reconstructed trajectory point of $T_i^t$ |
| $\hat{T}_i^t$ | predicted trajectory point of $T_i^t$ |
| $e_i^t$ | prediction error of $T_i^t$, i.e., $T_i^t - \tilde{T}_i^t$ |
| $C$ | error bounded codebook |
| $b_i^t$ | codeword index for $e_i^t$ |
| $\epsilon_1$ | spatial deviation threshold |
| $\epsilon_p$ | partition threshold for PPQ |
| $\epsilon_s$ | partition threshold for constructing index |
| $\tilde{T}_i^{t'}$ | reconstruction of $T_i^t$ considering CQC |

non-convex. For dynamic databases, $C$ needs to be incrementally updated with evolving $t$ values. In order to get the approximate solution, at time $t + 1$, if part of the prediction errors $\{e_i^{t+1}\}$ can not satisfy the threshold, the additional codewords are added to update $C$ to guarantee the boundary requirement continuously.

The reconstructed $\tilde{T}_i^t, \hat{T}_i^t$, is obtained where:

$$\tilde{T}_i^t = \hat{T}_i^t + C(b_i^t)$$

### Algorithm 1 Error-bounded Predictive Quantization

**Input:** $\{T_i^t\}$, $\epsilon_1$  
**Output:** $\{P_j |t\}$, $C$, $\{b_i^t\}$

1. $t = 1$
2. while $\{T_i^t\}$ is not empty do
3. Derive $P_j |t|$ based on Equation 1
4. $\hat{T}_i^t = \sum_{j=1}^{k} P_j |t| T_i^{t-k}$
5. $\{e_i^t\} = (T_i^t) - (\hat{T}_i^t)$
6. $C, \{b_i^t\} = \text{Incremental Quantizer} (\{e_i^t\}, C, \epsilon_1)$
7. $\tilde{T}_i^t = \hat{T}_i^t + C(b_i^t)$
8. $t = t + 1$
model them. However, as the role of $f_j$ is to capture the correlations between consecutive trajectory points, assigning trajectory points with similar autocorrelations to $N'_j$ can enable a more accurate prediction by $f_j$. In our setting, the correlation between $T^j_i$ and $T^j_{i-k+1}$ follows an autoregressive process of order $k$ (AR($k$)) [9, 34], where the current trajectory point ($T^j_i$) is linearly related to the lagged $k$ points ($T^j_{i-k+1}$). We derive the parameters of AR($k$) as $\{a_i^j\}$ and utilize them to quantify the lag-$k$ autocorrelation. Assigning trajectory points with similar $\{a_i^j\}$ to the same partition $N'_j$ allows $f_j$ to more effectively capture the correlations between consecutive trajectory points.

The partitioning process is repeated until all partitions satisfy Equations 7 and 8, for spatial and autocorrelation similarity, respectively. For the spatial proximity, the deviation between any point in $N_j$ and the centroid of $N_j$ should be less than $\varepsilon_p$, otherwise, $q$ increases until Equation 7 is satisfied.

$$\left\|T^j_i - \text{centroid}_S(N'_j)\right\|_2 \leq \varepsilon_p, \text{ for all } N'_j \in N^j, i \in N'_j \quad (7)$$

Similarly, for the autocorrelation similarity, the partitions satisfy Equation 8, where $a_i^j$ represents the lag-$k$ autocorrelation of $T^j_i$, and centroid$_A(N'_j)$ is the centroid of the autocorrelation of trajectory points in $N_j$.

$$\left\|a_i^j - \text{centroid}_A(N'_j)\right\|_2 \leq \varepsilon_p, \text{ for all } N'_j \in N^j, i \in N'_j \quad (8)$$

The setting of $\varepsilon_p$ is based on the size of the region that trajectories span (for spatial proximity), or the magnitude and distribution of autocorrelation coefficients (for autocorrelation similarity).

The computational complexity for $q$ partitions is $O(qmN_l)$ as shown in Lemma 1, where $N$ is the number of trajectory points, $m$ denotes the rounds of increasing $q$ to satisfy Equation 7 or 8, and $l$ represents the number of iterations to obtain the given number of partitions by K-means [28]. The complexity is proportional to $q$.

### 3.2.2 Incremental Temporal Partitioning

Consider the partitions at time $t$, i.e., $N^t = \{N'_1, ..., N'_q\}$. Instead of performing partitioning from scratch, an incremental temporal partitioning for time $t+1$ is performed with the following steps. First, every trajectory point at time $t+1$, i.e., $\{T^j_{i+1}\}$, is assigned to the same partition as $T^j_i$. Second, when a partition, $N_{i+1}^j$, does not satisfy the requirement for $\varepsilon_p$, a new partitioning is performed over trajectory points in $N_{i+1}^j$ until the resultant partition satisfies the requirement. Third, with $N^t_{i+1}$ and $N^{t+1}_{i+1}$, if $\left\|\text{centroid}_{S/A}(N_{i+1}^t) - \text{centroid}_{S/A}(N_{i+1}^{t+1})\right\|_2 \leq \varepsilon_p$, we merge $N^t_{i+1}$ to $N^{t+1}_{i+1}$ to avoid too many fragmented partitions. Specifically, for every partition $N_{i+1}^j$, we only merging at most once, as excessive merging might influence the preciseness of partitioning and the quantization performance. If there are $N'$ trajectory points at $t+1$ that do not satisfy the requirement for $\varepsilon_p$, $q'$ new partitions are generated via an $m'$ rounds of checking with Equation 7 or 8, then the computational complexity of the incremental temporal partitioning is $O(q'mN'l + q'q)$, which is presented in LEMMA 2. $q'$ is only relevant to the distribution of the $N'$ trajectory points. $N'$ gets smaller when the points among the consecutive timestamps are highly similar in autocorrelations or spatially close. In the worst case, when all the $N$ trajectory points at time $t+1$ do not satisfy the $q$ partitions at time $t$, i.e., $N = N'$, the computational complexity of incremental temporal partitioning becomes $O(qmN_l)$.

**LEMMA 1.** The computational complexity of partitioning $\{T^j_i\}$ into $q$ partitions (Section 3.2.1) is $O(qmN_l)$.

**Proof.** Let $q_i (i \in [1, m])$ be the number of partitions of the i-th round, which increases by $a$ at every round until all the partitions satisfy Equation 7 or 8. Hence, $q = q_m = ma$.

For the i-th round, the computational complexity is the same as partitioning $N$ trajectory points into $q_i$ clusters, i.e., $O(q_iN_l)$. Then, the overall computational cost is $aNI + ... + maNI = \frac{am(qm)}{2}N_l$, i.e., $O(qmN_l)$. □

**LEMMA 2.** The computational complexity of incremental temporal partitioning for $\{T^j_{i+1}\}$ is $O(q'mN'l + q'q)$.

**Proof.** According to LEMMA 1, the complexity of partitioning $N'$ trajectory points into $q'$ partitions is $O(q'mN'l)$. For $q'$ new partitions at time $t+1$, in the worst case, there will be $q + (q - (q' - 1)) = \frac{2(q'+q-q'}{2} = \frac{2}{2}$ computations to check if a new partition can be merged into the existing $q$ partitions. Hence, the overall complexity is $O(q'mN'l + q'q)$. □

### 4 LOCAL CODING IN ERROR-BOUNDED CODEBOOK

With the error-bounded codebook, the reconstructed value $(\hat{x}, \hat{y})$ is guaranteed to be within the circle $c_1$, as shown in Figure 2. While a small $\varepsilon_1$ is desirable for the accuracy of approximate query results, an excessively small $\varepsilon_1$ would degrade the effectiveness of quantization, both in terms of the efficiency of learning and the size of the codebook. Here, we present the coordinate quadtree coding (CQC), which encodes the spatial deviation between $(x, y)$ and $(\hat{x}, \hat{y})$, to reduce the information loss of the summary.

CQC consists of short binary codes that can be easily restored and utilized to quantity the lagged $T$ autocorrelation coefficients (for autocorrelation similarity). We derive the parameters of AR($q$) via an $m$ rounds of checking with Equation 7 or 8, then the computational complexity of the incremental temporal partitioning is $O(q'mN'l + q'q)$, which is presented in LEMMA 2. $q'$ is only relevant to the distribution of the $N'$ trajectory points. $N'$ gets smaller when the points among the consecutive timestamps are highly similar in autocorrelations or spatially close. In the worst case, when all the $N$ trajectory points at time $t+1$ do not satisfy the $q$ partitions at time $t$, i.e., $N = N'$, the computational complexity of incremental temporal partitioning becomes $O(qmN_l)$.

**LEMMA 1.** The computational complexity of partitioning $\{T^j_i\}$ into $q$ partitions (Section 3.2.1) is $O(qmN_l)$.

**Proof.** Let $q_i (i \in [1, m])$ be the number of partitions of the i-th round, which increases by $a$ at every round until all the partitions satisfy Equation 7 or 8. Hence, $q = q_m = ma$. For the i-th round, the computational complexity is the same as partitioning $N$ trajectory points into $q_i$ clusters, i.e., $O(q_iN_l)$. Then, the overall computational cost is $aNI + ... + maNI = \frac{am(qm)}{2}N_l$, i.e., $O(qmN_l)$. □

**LEMMA 2.** The computational complexity of incremental temporal partitioning for $\{T^j_{i+1}\}$ is $O(q'mN'l + q'q)$.

**Proof.** According to LEMMA 1, the complexity of partitioning $N'$ trajectory points into $q'$ partitions is $O(q'mN'l)$. For $q'$ new partitions at time $t+1$, in the worst case, there will be $q + (q - (q' - 1)) = \frac{2(q'+q-q'}{2} = \frac{2}{2}$ computations to check if a new partition can be merged into the existing $q$ partitions. Hence, the overall complexity is $O(q'mN'l + q'q)$. □

### 4 LOCAL CODING IN ERROR-BOUNDED CODEBOOK

With the error-bounded codebook, the reconstructed value $(\hat{x}, \hat{y})$ is guaranteed to be within the circle $c_1$, as shown in Figure 2. While a small $\varepsilon_1$ is desirable for the accuracy of approximate query results, an excessively small $\varepsilon_1$ would degrade the effectiveness of quantization, both in terms of the efficiency of learning and the size of the codebook. Here, we present the coordinate quadtree coding (CQC), which encodes the spatial deviation between $(x, y)$ and $(\hat{x}, \hat{y})$, to reduce the information loss of the summary.

CQC consists of short binary codes that can be easily restored and utilized to quantity the lagged $T$ autocorrelation coefficients (for autocorrelation similarity). We derive the parameters of AR($q$) via an $m$ rounds of checking with Equation 7 or 8, then the computational complexity of the incremental temporal partitioning is $O(q'mN'l + q'q)$, which is presented in LEMMA 2. $q'$ is only relevant to the distribution of the $N'$ trajectory points. $N'$ gets smaller when the points among the consecutive timestamps are highly similar in autocorrelations or spatially close. In the worst case, when all the $N$ trajectory points at time $t+1$ do not satisfy the $q$ partitions at time $t$, i.e., $N = N'$, the computational complexity of incremental temporal partitioning becomes $O(qmN_l)$.

**LEMMA 1.** The computational complexity of partitioning $\{T^j_i\}$ into $q$ partitions (Section 3.2.1) is $O(qmN_l)$.

**Proof.** Let $q_i (i \in [1, m])$ be the number of partitions of the i-th round, which increases by $a$ at every round until all the partitions satisfy Equation 7 or 8. Hence, $q = q_m = ma$. For the i-th round, the computational complexity is the same as partitioning $N$ trajectory points into $q_i$ clusters, i.e., $O(q_iN_l)$. Then, the overall computational cost is $aNI + ... + maNI = \frac{am(qm)}{2}N_l$, i.e., $O(qmN_l)$. □

**LEMMA 2.** The computational complexity of incremental temporal partitioning for $\{T^j_{i+1}\}$ is $O(q'mN'l + q'q)$.

**Proof.** According to LEMMA 1, the complexity of partitioning $N'$ trajectory points into $q'$ partitions is $O(q'mN'l)$. For $q'$ new partitions at time $t+1$, in the worst case, there will be $q + (q - (q' - 1)) = \frac{2(q'+q-q'}{2} = \frac{2}{2}$ computations to check if a new partition can be merged into the existing $q$ partitions. Hence, the overall complexity is $O(q'mN'l + q'q)$. □
Algorithm 2: Coordinate_QuadTree($c_1, g_s$)

\begin{enumerate}
\item Get the minimum rectangle $S$ covering $c_1$
\item $S$ is split into grids of equal size, $S_{g_s}$
\item $C_q = \emptyset$ # Coordinate quadtree
\item build_tree($S_{g_s}, C_q$)
\end{enumerate}

Function \texttt{build\_tree}($S_{g_s}, C_q$)

\begin{enumerate}
\item for all each sub-region $S_{g_s}$ in $[S_{g_s}, i]$ do
\item $s_x, s_y \leftarrow |S_{g_s}| \times s_x$ and $s_y$ are the number of grid cells of $S_{g_s}$ along the x- and y-axes respectively
\item if $s_{x,y} = 1$ and $\text{no\_padding}(S_{g_s})$ or $s_x, s_y = 0$ then
\item continue
\item $(S_{g_s}, i) \leftarrow \text{partition\_padding}(S_{g_s})$
\item $C_q.append(S_{g_s}, i)$
\item $\text{build\_tree}(S_{g_s}, i), C_q$
\end{enumerate}

the node represents, and the value over the edge is the quadrant that its parent node is located in.

\textbf{Definition 4.2.} (Coordinate Quadtree Coding) Given a coordinate quadtree, the coordinate quadtree coding (CQC) of a node $n_q$ is the values of the edges from the root node to node $n_q$ as well as the quadrant that $n_q$ is located in.

\section{4.1 Coordinate Quadtree Coding}

The process of building the coordinate quadtree, which is used as the basis for CQC, is summarized in Algorithm 2. The first step is to get the error space $c_1$ and find the minimum rectangle $S$ covering $c_1$ in Line 1. The second step is to divide $S$ into grid cells of equal size, $S_{g_s}$, in Line 2, where $g_s$ is the size of a cell. The third stage is to build the coordinate quadtree via Function \texttt{build\_tree}. Its stopping condition is either when the subspace is empty, or the input subspace is size one and without any padding grid cells, as shown in Line 3. For any $S_{g_s}$, its partitions are generated by \texttt{partition\_padding} in Line 4. An example is given in Figure 3. For simplicity, we omit $g_s$ and use $S'$ and $S_i$ to demonstrate the example. For function \texttt{partition\_padding}, if $S$ does not satisfy Line 1, we pad $S$ as $S'$ so that $S'$ can be partitioned into four equal partitions. To avoid conflicts of partitions at different rounds, we design specific padding rules for different quadrants. We have a $5 \times 5$ grid $S$ in Figure 3a, $S$ is expanded to a $6 \times 6$ grid $S'$ to obtain four equal size subspaces. According to Definition 4.1, the values $(-3,2), (2,2), (-3,-3)$ and $(2,-3)$ represent the size and quadrant for every subspace. The quadrants are encoded as $00, 01, 10$ and $11$ separately. As shown in Figure 3b, the subspace $S_1$ at quadrant $00$ is expanded towards the upper left as $S'_1$ which can be further partitioned. Similarly, the subspaces at quadrant $10$ and $11$, i.e., $S_3$ and $S_4$, are padded towards the bottom left and bottom right respectively, as shown in Figure 3c and Figure 3d. However, there is no need to pad $S_2$ as it can be directly partitioned into four subspaces of an equal size following Line 5. The full process of building a coordinate quadtree for $5 \times 5$ grids is shown in Figure 4a. The corresponding coordinate quadtree can be observed in Figure 4b. The value in every square corresponds to the coordinate of a subspace it denotes, which is denoted as $SC$. “X” denotes the empty padding grid. According to Definition 4.2, the CQC for the $n_1$ in Figure 4 is 001110.

For a CQC, the real coordinate is the sum of the values of the nodes it visits, as shown in Equation 9.

$$c_{cqc} = \sum_{j=1}^{m} \frac{1}{2} SC'_j$$

(9)

$$SC' = \begin{cases} 
\frac{|x| + |y|}{2}, & \text{sgn}(x) \cdot \text{sgn}(y) \\
\frac{|x| - |y|}{2}, & \text{otherwise}
\end{cases}$$

(10)

where $SC'$ denotes the SC of the padded subspace. $SC'$ is obtained by Equation 10. For example, in Figure 4b, a $SC$ is $(-3,2)$, its $SC'$ is $(-4,4)$ by Equation 10. According to Equation 9, the real coordinate for $n_1$ is $(-\frac{1}{2}, \frac{1}{2})$.

If $S_{g_s}$ cannot be partitioned into four subspaces with the same number of grid cells, the traditional approach for quadtrees [37] would extend $S_{g_s}$ and maintain the empty grids. However, the padding process guarantees to produce four equally sized partitions. We utilize the coordinate of the subspace, to keep track of the real size of every subspace and the relative displacement of grid cells among the subspaces at different levels, and make sure...
When \( \epsilon_2 \) and \( g_s \) are given, \((x, y)\) is fixed at the center cell of \( S_g \), i.e., \( n_2 \) in Figure 4. Its CQC is represented as \( cqc2 \). With \( cqc1 \) and \( cqc2 \), the reconstructed trajectory point for \((x, y)\) is obtained as:

\[
(x', y') = (x, y) + g_s \cdot (cqc1 - cqc2)
\]

where \( cqc1 \) and \( cqc2 \) are CQC of \((x, y)\) and \((x, y)\), obtained by Equation 9.

As shown in Figure 4b, \((x, y)\) is denoted as \( n_2 \). \((x, y)\) is represented as \( n_1 \). According to Equation 9, the real coordinates of \((x, y)\) and \((x, y)\) at the \( S_g \) are \( ccqc1 = (-\frac{1}{2}, -\frac{1}{2}) \) and \( ccqc2 = (-\frac{1}{2}, \frac{1}{2}) \), respectively. Then, when \((x, y)\) is known, the reconstructed point, \((x', y')\), can be obtained using Equation 10. Given \( \epsilon_1 \) and \( g_s \), a unified and fixed coordinate quadtree is obtained. The coordinate quadtree is stored as a template to recover the position with the given CQC.

For any original trajectory points \((x, y)\), they have the same CQC, i.e., \( cqc1 \), as they are always located at the center of the same grid of \( S_g \). Hence, only the coding of \((x, y)\), i.e., \( cqc2 \), is stored for every trajectory point sample. \((x, y)\) is recovered from the summary produced by our approach. It is easy to prove that deviation has been reduced within \( \frac{\sqrt{2}}{2} g_s \) as shown in Lemma 3.

**Lemma 3.** Given the grid size \( g_s \), the trajectory point \((x, y)\), the error of the accurate reconstructed trajectory \((x', y')\) does not exceed \( \frac{\sqrt{2}}{2} g_s \), i.e., \( 0 \leq \|(x, y) - (x', y')\|_2 \leq \frac{\sqrt{2}}{2} g_s \).

### 4.2 Trajectory Reconstruction with CQC

Given the grid size \( g_s \), the trajectory point \((x, y)\), the error of the accurate reconstructed trajectory \((x', y')\) does not exceed \( \frac{\sqrt{2}}{2} g_s \), i.e., \( 0 \leq \|(x, y) - (x', y')\|_2 \leq \frac{\sqrt{2}}{2} g_s \).

### 5 ONLINE QUERYING OVER QUANTIZED TRAJECTORIES

The last part of PPQ-trajectory is the organization of quantized spatio-temporal data for online querying to obtain the candidate set without the reconstruction of all trajectories. A simple spatio-temporal query example is to search vehicles that travel through location \((x, y)\) at time \( t \), and estimating their next \( l \) positions. We focus on two commonly used spatio-temporal queries, as presented later in the paper.

Since the parameters in the system \((P_{\{1\}}, C, \{b^i\}, \text{CQC})\) are enough to reproduce any trajectory, the naive solution is to reconstruct the trajectories from time 1 to \( t \) and return the trajectories that match the query conditions. In Section 5.1, we propose a temporal partition-based organization to enable direct access to the relevant trajectory IDs at time \( t \), and get their next \( l \) positions in an online manner. In Section 5.2, we illustrate the spatio-temporal query process over quantized trajectories, and a local search strategy motivated by CQC is introduced to achieve the accurate query.
cell. To reduce the storage cost, we compress trajectory IDs mapped to the grid cell by delta encoding and Huffman codes, following the approach in the other works [19, 22, 42]. The PI at time \( t \), i.e., \( pi_t \), is returned as shown in Line 12.

We avoid building PI from scratch for every timestamp to efficiently maintain dynamic trajectories. For example, at time \( t_e \), \( T^{t_e} \) are indexed by \( pi_{t_e} \). At time \( t_e \), part of \( pi_{t_e} \) might be reused for \( T^{t_e} \), as the distributions among consecutive timestamps might not change sharply.

**Definition 5.1.** (Trajectory Region Density (TRD)) Given \( T^t \) and its PI \( pi^t=\{R_{1,g},...,R_{n,g}\} \), for \( R_{i,g}\), its TRD is \( d(R_{i,g}, t) = \frac{N_{R_{i,t}}}{|R_{i,g}|} \), where \( |R_{i,g}| \) denotes the size of rectangle \( R_i \). \( N_{R_{i,t}} \) is the number of trajectories indexed by \( R_{i,g} \) at time \( t \).

The definition of TRD quantifies the occupancy rate of subregions, providing the basis of building the temporal index flexibility. According to Definition 5.1, we compute the average dropping rate (ADR) of TRD by Equation 12, to measure reusing the previous index \( pi_t \) or constructing a new index. For example, at time \( t_s \), its index is \( pi_s = \{R_{1,g},...,R_{n,g}\} \). The TRD of subregion \( R_{i,g} \) at time \( t_s \), i.e., \( d(R_{i,g}, t_s) \), is obtained. For trajectory points at time \( t_e \), the new TRD \( d(R_{i,g}, t_e) \) is computed. For \( R_{i,g} \), the dropping rate of its TRD from \( t_s \) to \( t_e \) can be obtained with Equation 13. For \( R_{i,g} \), if the dropping rate of its TRD, i.e., \( R_{i,g} \), exceeds the threshold \( \epsilon_e \), then it counts for ADR since the occupancy rate of \( R_{i,g} \) drops too much, which is achieved by Equation 14.

\[
ADR(t_s, t_e, \epsilon_e) = \frac{\sum_{i=1}^{n} h_1(R_{i,g}, t_s, t_e, \epsilon_e)}{n} (12)
\]

\[
h_1(R_{i,g}, t_s, t_e, \epsilon_e) = \frac{d(R_{i,g}, t_s) - d(R_{i,g}, t_e)}{d(R_{i,g}, t_s)} (13)
\]

\[
h(x, \epsilon_e) = \begin{cases} 1 & x < 0 \text{ and } |x| > \epsilon_e \vspace{1em} \\ 0 & \text{otherwise} \end{cases} (14)
\]
We now present how PPQ-trajectory answers spatio-temporal queries. A smaller $\epsilon_d$, the operation of "Re-build" will be more frequent due to the strict constraints for ADR. An example is presented in Figure 5, with $\epsilon_c = 0.5$, $\epsilon_d = 0.5$, $|R_{i,g_k}| = 1$ (in Definition 5.1), and $d(R_{i,g_k}, t) = N_{R_{i,j}}$, i.e., the number of nodes in $R_{i,j}$ at time $t$. As shown in Figure 5b, TRD at $t + 1$ has changed, i.e., $d(R_{i,g_k}, t + 1)$, and according to Equation 12–14, we get $ADR(t, t + 1, \epsilon_c) = 0.75 > \epsilon_d$, i.e., the average dropping rate of TDR exceeds the threshold, which means the PI at $t$ can not be further reused to index trajectories at time $t + 1$, hence a PI is rebuilt. However, in Figure 5c, $ADR(t, t + 1, \epsilon_c) = 0.25 < \epsilon_d$, then we can hold the PI at $t$, i.e., $\{R_{i,j}, R_{i,g_k}, R_{j,g_k}, R_{k,g_k}\}$, and only build a new PI for trajectory points that are not covered by PI at $t$, i.e., indexing the uncovered trajectory points $T_{d+1}$ in $R_{i,g_k}$ which is "Insertion".

The merits of TPI are two-fold. First, based on the dynamic trajectory density, spatio-temporal trajectories are indexed as a set of periods, which lowers the frequency of partitioning the spatial region. Second, with TPI, the accuracy of the partitions within a certain period is guaranteed by the measurement of ARD.

For disk-resident data, the trajectory points within a time period can be written into several pages and the corresponding part of the summary, i.e., $\hat{c}_k = \{\hat{p}_j[t], C, \{\hat{h}'\}, \text{CQC}\}$, is assigned to the corresponding page. A lightweight index for the assigned page number is used to record the assigned pages for the trajectory points of period $j$, and the corresponding summary, i.e., $\{\text{period}_j\}$, starting page number, relative page number).

### 5.2 Spatio-temporal Query Processing

We now present how PPQ-trajectory answers spatio-temporal queries. We illustrate our approach using Spatio-temporal Range Query (STRQ) and Trajectory Path Query (TPQ).

**Definition 5.2. (Spatio-temporal Range Query (STRQ))** Given time $t$ and location $(x, y)$, STRQ retrieves trajectories which are located at the grid cell that $(x, y)$ is in at time $t$.

For STRQ, given a query $(x, y, t_1)$, if $t_1 \in \text{period}_j$, the sub-regions of $p_{ij}$ are obtained, e.g., $R_{i,g_k}(x, y)$ is mapped to a grid of $R_{i,g_k}$, then a list of trajectory IDs at time $t_1$ is returned.

**Definition 5.3. (Trajectory Path Query (TPQ))** Given time $t$, location $(x, y)$ and the path duration $l$, the trajectory IDs of the STRQ of $(x, y, t)$ are first retrieved, then their sub-trajectories at the time interval $[t,t+l]$ are returned.

For TPQ, given time $t$, location $(x, y)$ and path duration $l$, the list of trajectory IDs is first returned from STRQ by searching $(x, y, t)$, the next $l$ positions of the retrieved trajectories are then directly reproduced by the indexed summary.

**Local Search using CQC.** For a trajectory point $(x, y)$, its accurate reconstructed trajectory point is $(\hat{x}', \hat{y}')$. According to Lemma 3, $(\hat{x}', \hat{y}')$ might be any point within a circle of which radius is $\sqrt{2}g_c$ and $(x, y)$ is the center. With the CQC, the deviation has been narrowed to $\frac{\sqrt{2}}{2}g_c$, which is a smaller distance. In order to further improve the accuracy of queries using the summary, we introduce a local search strategy. There are two situations for the local search.

1. When $\frac{\sqrt{2}}{2}g_c > g_c$, as shown in Figure 6a, when all the grid cells $g_c$ that are covered by the circle are scanned, the full actual result can be retrieved successfully. (2) When $\frac{\sqrt{2}}{2}g_c \leq g_c$, as shown in Figure 6b, the worst case is that $(\hat{x}', \hat{y}')$ falls out of the grid cell $g_c$ as $(x, y)$ happens to be adjacent to the border of the grid cell. Hence, in order to guarantee that the actual result is retrieved, the quantized trajectory points of which distance to $(x, y)$ is less than $\frac{\sqrt{2}}{2}g_c$ in the grid cells that are adjacent to the grid cell $(x, y)$ mapped to are all scanned. Specifically, the second case is more general, i.e., $g_c$ is smaller, as the aim of introducing $g_c$ is to further reduce the information loss. For $g_c$, it serves for the index part, i.e., TPI, hence, it is usually larger.

With the local search, the returned candidate list contains all the trajectory IDs for STRQ, which makes the recall 1. However, the candidate list may include the trajectory IDs of which true position is at the boundary of adjacent cells of $g_c$ while mapped to $g_c$. The $(\hat{x}', \hat{y}')$ whose distance to $(x, y)$ is larger than $\frac{\sqrt{2}}{2}g_c$ has been filtered out according to Lemma 3. The precision for STRQ can be improved to 1 by accessing the original trajectory of the candidate list of which size is relatively small due to the accuracy of $(\hat{x}', \hat{y}')$.

### 6 EXPERIMENTAL EVALUATION

We conduct a range of experiments to evaluate the effectiveness of PPQ-trajectory and compare it with a variety of alternative approaches. Our methods and all the alternative approaches are implemented in Matlab R2019a. All experiments are executed on an Ubuntu 19.04 with an Intel i5-8500 3.00 GHZ GPU and 31GB RAM.

#### 6.1 Experimental Setting

**Datasets.** The experiments are performed on the publicly available trajectory datasets, Porto [11] and GeoLife [46]. We select the trajectories with the length being at least 30. The selected Porto dataset contains 1.2 million trajectories, with 74.3M trajectory points and the longest trajectory consisting of 3881 location points. The GeoLife dataset retains 17,932 trajectories with the maximum length of 92,645, containing 24.8M trajectory points.

**Compared Methods.** We implemented the extended versions of alternative methods from the literature: Product Quantization [19], Residual Quantization [8], REST [44] and TrajStore[10]. We compare with one of the variations of REST, i.e., trajectory redundancy reduction, which was shown to perform best in their work [44]. We also test simple baselines as well as variants of PPQ-trajectory to quantify the improvements of each step.
Product Quantization and Residual Quantization are popularly used for approximate nearest neighbor (ANN) queries. However, they normally do not offer an effective index structure to support pruning for efficient querying. For fairness, we extended these methods with our indexing approach used in PPQ-trajectory.

For REST to work properly, the dataset needs to contain a highly repeating set of patterns between the trajectory set for building the reference set and the matched trajectory. This is not the case for most of the real-world data, including the datasets we use. To be able to compare with REST, we constructed another dataset. We first randomly selected 20,000 trajectories from Porto dataset, then for every trajectory we created four more similar trajectories by down-sampling and adding noise following the procedure in [23]. This process gives a dataset with 100,000 trajectories that is suitable for REST. Specifically, 2,000 trajectories are randomly selected for compression, we name this dataset as sub-Porto, while other trajectories are used to build a reference set for REST.

TrajStore builds an index with the spatial regions and recursively updates the spatial index by merging, splitting or appending. To align with the experimental setting, we implemented TrajStore to be able to get streaming trajectory points as input, and dynamically build the spatial index with time increasing.

We also implemented different variants of PPQ-trajectory to understand the effect of its building blocks. PPQ-S uses the spatial proximity for partitioning, PPQ-A uses the autocorrelations based similarities. PPQ-S-basic and PPQ-A-basic use the quantizers but not CQCs. As other baselines, we include comparisons with E-PQ, and also with a basic version of PPQ-trajectory, by skipping the prediction part, and name it as Q-trajectory. Parameter Settings. The default quantization deviation threshold is $\epsilon_1 = 0.001$, which is $\epsilon_1^M \approx 111$ meters under the geographic coordinate system [6]. In the following experiments, we directly use $\epsilon_1^M$ to describe the comparative study. For the partition threshold $\epsilon_p$, its setting varies on the autocorrelation and spatial-proximity-based partitions. For the spatial proximity-based solution, $\epsilon_p$ defaults to 0.1 for Porto and for Geolife. In the case of autocorrelation similarity, $\epsilon_p$ defaults to be 0.01 for both datasets. For TPI and PI, the grid cell size $g_c$ is set to 100m, $\epsilon_d$ defaults to 50m, which denotes the size of the grid cell for CQC. The threshold of the dropping rate of TRD, $\epsilon_d$ is defaulted to be 0.5. The default setting of the threshold of ADR, i.e., $\epsilon_d$, is 0.5. $\epsilon_s$ defaults to 0.1, which represents the partition threshold for constructing index.

6.2 Query Performance

6.2.1 Spatio-temporal Range Query. The quality of the approximate results for STRQ is evaluated in terms of precision and recall. The precision is the ratio of the correctly retrieved trajectory IDs to the returned candidate list, and the recall is the ratio of the correctly retrieved trajectory IDs to all the trajectory IDs that match the query. We also measure the MAEs of the summaries over the datasets, i.e., the mean absolute errors between the reconstructed trajectory points and the original trajectory points.

For STRQ, we learn C independently for every timestamp guaranteeing the same number of codewords is given to trajectory points at the same time across all methods. We randomly select 10,000 queries. The comparative results are summarized in Table 2. For MAE, the PPQ-trajectory significantly performs better than other methods. As the time increases, PPQ-trajectory is able to gradually quantize a narrower range, whereas, residual and product quantization do not improve over time. The summary process of Trajstore cannot start until the spatial index has been updated with trajectory points of all the timestamps. To ensure fairness, the codewords are assigned in proportion to the number of trajectory points for every spatial cell of TrajStore.

With the same number of bits, PPQ-trajectory obtains significantly higher recall and precision values. For Geolife, autocorrelation-based partitioning (in PPQ-A and PPQ-A-basic) helps to achieve smaller MAEs compared to the spatial proximity based solution (PPQ-S and PPQ-S-basic), while PPQ-S-basic outperforms PPQ-A-basic on Porto dataset. We observe that autocorrelation similarity possesses some advantages upon capturing correlations and obtaining a narrower dynamic range of prediction errors. This result provides a useful insight also for other partitioning tasks and applications of spatio-temporal data, which is discussed in Section 3.2.1.

6.2.2 Trajectory Path Query. TPQ involves querying timestamps and trajectory IDs of the STRQ results and reconstructing their next 10–50 trajectory points. According to Definition 5.3, the retrieved sub-trajectories of TPQ depends on the returned trajectory IDs of the STRQ. Different methods might retrieve sub-trajectories for different trajectory IDs, as observed in their different recall and precision values presented in Table 2. For fairness, we select 10,000 same trajectory IDs for all the methods to measure the MAEs of the retrieved sub-trajectories by comparing each to the corresponding original sub-trajectory.

The comparative results are summarized in Table 3. The MAEs for the sub-trajectories increase with the increasing TPQ lengths, because more spatial deviations are accumulated when querying longer sub-trajectories. The PPQ-trajectory and E-PQ significantly perform better than other methods, while the MAEs of E-PQ are smaller than that of PPQ-A-basic and PPQ-S-basic. The MAEs of Q-trajectory, residual quantization and product quantization increase significantly with the increasing query length, because their MAEs on the datasets have been extremely large (Table 2). We notice the MAEs of TrajStore over Porto get relatively large with $l$ increasing while its MAE on the full dataset is smaller (Table 2). The codewords are assigned in proportion to the number of trajectory points for
every spatial cell of TrajStore (Section 6.2.1), hence, a larger spatial cell with a smaller number of trajectory points scattering in will be assigned a smaller number of codewords, then there will be larger deviations of summarizing the trajectory points of this spatial cell, even though the average deviations over all the spatial cells are smaller.

6.3.1 Summary Efficiency. We evaluate the running times of generating the summary for different solutions with spatial deviations as 200m, 400m, 600m, 800m and 1000m. According to Lemma 3, the spatial deviation of PPQ-A and PPQ-S are \(\frac{1}{2} e_1\). In the experiment, we set \(e_1^M = 2g_1\) for PPQ-A and PPQ-S. For the other methods, the spatial deviation of their summary is simply determined by \(e_1^M\).

| Dataset | MAE(\(\mu\)) | Precisions | Recall | MAE(\(\mu\)) | Precisions | Recall |
|---------|---------------|------------|-------|---------------|------------|-------|
| PPQ-A   | 18.35         | 1.000      | 1.000 | 18.35         | 1.000      | 1.000 |
| PPQ-A-basic | 51.92         | 0.951      | 0.948 | 6.17          | 0.987      | 0.987 |
| PPQ-S   | 25.30         | 1.000      | 1.000 | 7.19          | 1.000      | 1.000 |
| PPQ-S-basic | 44.41         | 0.944      | 0.939 | 14.72         | 0.976      | 0.976 |
| E-PQ    | 76.60         | 0.931      | 0.926 | 15.06         | 0.962      | 0.961 |
| Q-trajectory | 1752.29      | 0.425      | 0.427 | 29101         | x          | x     |
| TrajStore | 152.13        | 0.917      | 0.919 | 617.76        | 0.8355     | 0.8547 |

Table 2: Quality of summaries and STRQ evaluation

6.3.2 Filtering for Exact Match Queries. We now present the average ratios of trajectories visited when the summary is used as an index for exact match queries. After pruning irrelevant data, only a set of candidates is accessed. We randomly select 10,000 queries, and the average ratios of trajectories visited in their second step are presented. To ensure fairness, we learn \(C\) independently for every spatial cell of TrajStore (Section 6.2.1), hence, a larger spatial cell with the trajectory points of all timestamps. Hence, for TrajStore, we cannot fairly summarize the trajectory points of every timestamp independently with the fixed size \(C\). The same applies to PPQ-S. The building time is summarized in Table 5, which gradually increases for most of the methods. However, the MAEs of PPQ-A and PPQ-S do not strictly decline with the size of \(C\) increasing, because their spatial deviation is not fully decided by the quality of \(C\), but also slightly influenced by CQCs.

| Dataset | Predictive time (\(\mu\)) | 10 | 20 | 30 | 40 | 50 |
|---------|---------------------------|----|----|----|----|----|
| PPQ-A   | 0.046                      | 0.081 | 0.111 | 0.136 | 0.158 | 0.111 |
| PPQ-A-basic | 0.357                     | 0.657 | 0.935 | 1.194 | 1.437 | 0.073 |
| PPQ-S   | 0.160                      | 0.295 | 0.402 | 0.491 | 0.566 | 0.019 |
| PPQ-S-basic | 0.338                 | 0.623 | 0.890 | 1.140 | 1.374 | 0.135 |
| E-PQ    | 0.068                      | 0.119 | 0.162 | 0.198 | 0.249 | 0.031 |
| Q-trajectory | 24.90                    | 44.55 | 62.02 | 77.54 | 91.75 | 190.2 |
| TrajStore | 5.665                     | 10.32 | 14.66 | 18.56 | 22.09 | 14.99 |

Table 3: MAE against different lengths of TPQ (1.0e-3 m)

6.3.3 Building Time Efficiency. We compare the performance of PPQ-A and PPQ-S, which need fewer iterations to obtain the summary. For Porto dataset, the running time of our solution is up to 25 times faster than residual quantization, product quantization, and TrajStore. In our solution, the dynamic range of the index is built with the trajectory points of all timestamps. Hence, for PPQ-A and PPQ-S is larger than that of PPQ-A-basic and PPQ-S-basic, respectively on both datasets, because for the same spatial deviation, the setting of \(e_1^M\) for PPQ-A and PPQ-S is larger than that of PPQ-A-basic and PPQ-S-basic, which needs fewer iterations to obtain the summary. For Porto dataset, the running time of our solution is up to 25 times faster than residual quantization, product quantization, Q-trajectory, and TrajStore, while E-PQ is faster than PPQ-A when the spatial deviation is larger. The running time of E-PQ is comparatively high on Porto, which is even larger than residual quantization. It shows the E-PQ cannot work as efficiently as our solution on the large datasets, because E-PQ executes one prediction on the whole datasets, the prediction errors will be larger and need

| Dataset | Predictive time (\(\mu\)) | 10 | 20 | 30 | 40 | 50 |
|---------|---------------------------|----|----|----|----|----|
| PPQ-A   | 0.019                      | 0.019 | 0.019 | 0.019 | 0.019 | 0.019 |
| PPQ-A-basic | 0.046                     | 0.081 | 0.111 | 0.136 | 0.158 | 0.111 |
| PPQ-S   | 0.022                      | 0.032 | 0.026 | 0.024 | 0.021 | 0.017 |
| PPQ-S-basic | 0.039                 | 0.042 | 0.042 | 0.042 | 0.042 | 0.042 |
| E-PQ    | 0.112                      | 0.057 | 0.042 | 0.028 | 0.028 | 0.028 |
| Q-trajectory | 18.52                    | 19.86 | 20.13 | 20.41 | 20.70 | 21.00 |
| TrajStore | 775.52                    | 775.52 | 775.52 | 775.52 | 775.52 | 775.52 |

Table 4: Average ratio of trajectories visited and MAE against different sizes of \(C\) (1.0e-3 m)

The building time is summarized in Table 5, which gradually decreases as the spatial deviation increases. This because the quantization process finishes with fewer iterations when the error is larger. The running times of PPQ-trajectory are much smaller than those of Q-trajectory, residual quantization, product quantization, and TrajStore. In our solution, the dynamic range of the prediction errors that needs to be quantized is decreasing with time \(t\) evolving, hence its running time is smaller. PPQ-A and PPQ-S are more efficient than PPQ-A-basic and PPQ-S-basic, respectively on both datasets, because for the same spatial deviation, the setting of \(e_1^M\) for PPQ-A and PPQ-S is larger than that of PPQ-A-basic and PPQ-S-basic, which needs fewer iterations to obtain the summary. For Porto dataset, the running time of our solution is up to 25 times faster than residual quantization, product quantization, Q-trajectory, and TrajStore, while E-PQ is faster than PPQ-A when the spatial deviation is larger. The running time of E-PQ is comparatively high on Porto, which is even larger than residual quantization. It shows the E-PQ cannot work as efficiently as our solution on the large datasets, because E-PQ executes one prediction on the whole datasets, the prediction errors will be larger and need
more iterations to satisfy the spatial deviation requirement. For Geolife dataset, the running time of our solution is 4-78 times faster than residual quantization, product quantization, Q-trajectory, and TrajStore, while E-PQ is slightly faster than PPQ-trajectory for some spatial deviations. The running time of residual quantization, product quantization and Q-trajectory drops quickly with the spatial deviation increasing. However, their running time is extremely high when the spatial deviation is 200m, as the time span of Geolife is relatively large, which needs even more iterations to summarize.

The running time of TrajStore is extremely high for all spatial deviations. Its running time includes both the time of building the spatial index and compression for every spatial cell, because TrajStore depends on the spatial index to conduct the summarization, and the process of building the index is time-consuming due to the frequent merging, splitting, and appending operations.

6.3.2 Dynamic Data Organization. In this section, we analyze the proposed partition-based index (PI) and temporal PI (TPI) on \( T_s \) with different \( \epsilon_c \) and \( \epsilon_d \), in terms of building time cost, the number of partitioned time periods, "Insertion" and "Re-build".

Table 7 reports that as \( \epsilon_c \) increases, the index size gradually decreases, since a higher \( \epsilon_c \) provides a higher tolerance of reusing the previous structure by performing "insertions". Similar results are observed for \( \epsilon_d \) in Table 8, i.e., a higher \( \epsilon_d \) allows a PI reused for more timestamps.

6.3.3 Temporal Partitioning Efficiency. In this section, we evaluate the efficiency of the incremental temporal partitioning (Section 3.2.2), and analyze how the number of partitions change with time, with respect to different \( \epsilon_p \) values.

Figure 7 illustrates that the running time of the temporal partitioning component reduces as \( \epsilon_p \) increases, since a smaller number of partitions is produced when \( \epsilon_p \) gets larger. In Figure 8, we also present the number of partitions that is maintained will gradually get stable with time increasing. For example, in Figure 8a, we get the maximum number of partitions, 83, on Porto dataset at \( t = 1072 \).

6.4 Compression Ratio

In this section, the compression ratios of different methods are measured for different values of spatial deviation, following the same parameters setting as Section 6.3.1. The comparative results are presented in Figure 9. For the Porto dataset, our solution outperforms Q-trajectory, residual quantization, and product quantization. The compression ratios of PPQ-A-basic and PPQ-S-basic slightly outperform PPQ-A and PPQ-S, respectively, because PPQ-A and PPQ-S need additional space to store CQC. The size of codebooks of E-PQ and TrajStore is up to 11 and 27 times larger, respectively, than PPQ-trajectory (Table 6), however, they achieve compression ratios that are higher than PPQ-trajectory, because we need additional space for multiple partitions.

For example, in Figure 8a, we get the maximum number of partitions, 83, on Porto dataset at \( t = 1072 \).
Figure 9: Compression ratio against different spatial deviation

Table 9: Disk-based index performance

| Dataset       | Index Size(MB) | No.I/Os | Response Time(s) | Building Time(s) |
|---------------|----------------|---------|------------------|------------------|
| TPI           | 837.4          | 233.5   | 2230             | 24               | 283               | 4665              | 2888              |
| PI            | 870.5          | 271.9   | 338              | 18               | 1572              | 32009             |
| TrajStore     | 837.4          | 338     | 35233            | 147              | 4244              | 24372             |

6.5 Further Comparison with TrajStore

In this section, we provide disk-based comparisons of temporal partition-based index (TPI) with TrajStore in terms of the index size, query response times, the number of I/Os during queries, and building times. The index of TrajStore is built on the raw trajectory points for fairness, we align disk-based TPI with TrajStore to directly build index over the raw trajectory points in accordance with the end of Section 5.1. We followed the same process in the TrajStore[10], bounding the data on disk and setting the page size as 1MB. We randomly select 10,000 spatio-temporal queries and sort them in the order of their starting times. The parameters for TPI are $\varepsilon_d = 0.8$ and $\varepsilon_c = 0.5$.

These experimental results are presented in Table 9. The index size and building time of PI are larger than that of TPI and TrajStore for both datasets, while its response time and I/Os are the smallest among the three methods on both datasets. For Porto, the size of TPI is the same as TrajStore, while the index size of TrajStore for Geolife is slightly smaller than that of TPI. However, TPI continuously outperforms TrajStore in terms of the number of I/Os and query response times. Given a query, TPI can quickly target the relevant period $j$ and filter out more irrelevant trajectory points in terms of time. However, for TrajStore, the quadtree-based index structure is shared by all timestamps, and a spatial cell can include trajectory points of a large time range, which might be stored on different pages. Hence, given a spatio-temporal query, for TrajStore several pages are likely to be visited, this is why the number of I/Os for TrajStore is over the number of queries we evaluate on.

Building of TPI is more efficient than TrajStore. TrajStore recursively updates the spatial index by merging, splitting or appending with trajectory points updated. However, when we get updates for TPI, it is only relevant to the trajectory points within a smaller range of timestamps, i.e., a period $j$.

7 CONCLUSIONS

We presented PPQ-trajectory which generates and maintains an error-bounded summary for large-scale trajectory data analytics. A partition-wise predictive quantizer (PPQ) for spatio-temporal data is designed, which involves a spatial proximity and autocorrelation based partitioning, followed by a local coding. A temporally-quantized data organization is developed to process spatio-temporal queries efficiently. The query performances, building times, and compression capabilities of PPQ-trajectory significantly outperform other solutions in most of the experiments. As a future work, the quantization based approach can be enhanced to consider dynamic traffic conditions, and utilize machine learning to more accurately predict trajectory points and generate a more compact summary.
REFERENCES

[1] Fatih Altiparmak, Ertem Tuncel, and Hakan Ferhatosmanoglu. 2007. Incremental maintenance of online summaries over multiple streams. *IEEE Transactions on Knowledge and Data Engineering* 20, 2 (2007), 216–229.

[2] Richard E Bellman and Stuart E Dreyfus. 2015. *Applied dynamic programming*. Vol. 2050. Princeton university press.

[3] Yuhan Cai and Raymond Ng. 2004. Indexing spatio-temporal trajectories with Chebyshov polynomials. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. 599–610.

[4] Zhiheng Li, Ken CK Lee, Baihua Zheng, Wang-Chien Lee, Dik Lee, and Xufa Wang. 2010. Kr-tree: An efficient index for geographic document search. *IEEE Transactions on Knowledge and Data Engineering* 23, 4 (2010), 585–599.