# Quantifying Point Contributions: A Lightweight Framework for Efficient and Effective Query-Driven Trajectory Simplification

Yumeng Song
Northeastern Univ., China
songym@stumail.neu.edu.cn

Yu Gu*
Northeastern Univ., China
guyu@mail.neu.edu.cn

Tianyi Li*
Aalborg Univ., Denmark
tianyi@cs.aau.dk

Yushuai Li
Aalborg Univ., Denmark
yusli@cs.aau.dk

Christian S. Jensen
Aalborg Univ., Denmark
csj@cs.aau.dk

Ge Yu
Northeastern Univ., China
yuge@mail.neu.edu.cn

## ABSTRACT

As large volumes of trajectory data accumulate, simplifying trajectories to reduce storage and querying costs is increasingly studied. Existing proposals face three main problems. First, they require numerous iterations to decide which GPS points to delete. Second, they focus only on the relationships between neighboring points (local information) while neglecting the overall structure (global information), reducing the global similarity between the simplified and original trajectories and making it difficult to maintain consistency in query results, especially for similarity-based queries. Finally, they fail to differentiate the importance of points with similar features, leading to suboptimal selection of points to retain the original trajectory information.

We propose MLSimp, a novel Mutual Learning query-driven trajectory simplification framework that integrates two distinct models: GNN-TS, based on graph neural networks, and Diff-TS, based on diffusion models. GNN-TS evaluates the importance of a point according to its globality, capturing its correlation with the entire trajectory, and its uniqueness, capturing its differences from neighboring points. It also incorporates attention mechanisms in the GNN layers, enabling simultaneous data integration from all points within the same trajectory and refining representations, thus avoiding iterative processes. Diff-TS generates amplified signals to enable the retention of the most important points at low compression rates. Experiments involving eight baselines on three databases show that MLSimp reduces the simplification time by 42%–70% and improves query accuracy over simplified trajectories by up to 34.6%.

**PVLDB Reference Format:**
Yumeng Song, Yu Gu, Tianyi Li, Yushuai Li, Christian S. Jensen, and Ge Yu. Quantifying Point Contributions: A Lightweight Framework for Efficient and Effective Query-Driven Trajectory Simplification. PVLDB, 18(2): 453 - 465, 2024.
doi:10.14778/3705829.3705858

**PVLDB Artifact Availability:**
The source code, data, and/or other artifacts have been made available at https://github.com/yumengs-exp/MLSimp.

*Corresponding authors

Proceedings of the VLDB Endowment, Vol. 18, No. 2 ISSN 2150-8097.
doi:10.14778/3705829.3705858

## 1 INTRODUCTION

The widespread use of mobile and location-aware devices generates large volumes of GPS trajectory data. Efficient compression of this data [13, 23, 25, 34] is attractive as it reduces storage and transmission costs. Trajectory simplification [42], which focuses on retaining only essential points, provides an efficient space reduction with acceptable information loss.

Most trajectory simplification methods [14, 31, 44], such as those using Synchronized Euclidean Distance (SED) [32, 33, 35], Perpendicular Euclidean Distance (PED) [26, 27, 31], Direction-aware Distance (DAD) [17, 18], and Speed-aware Distance (SAD) [33], evaluate the importance of GPS points based on error margins. They are Error-Driven Trajectory Simplification (EDTS) methods. Such methods often fail to accommodate common queries. An experimental report [49] finds that simplification methods using the DAD error, e.g., [28, 29], result in nearly a 50% drop in performance on range queries. Simplification methods using the PED and SED errors, e.g., [7, 14, 31], lead to a performance decline of up to 57% on $k$NN queries and up to 33% on similarity queries.

Query-Driven Trajectory Simplification (QDTS) [45] has been proposed to address the above limitations. QDTS aims to ensure that the results of queries on simplified trajectories match those on the original trajectories closely across various query types, while also achieving low compression rates. Motivated by these advantages, we focus on batch mode QDTS. In this mode [14, 31, 41, 45], all trajectories are stored in the database and simplified offline once. The resulting simplified trajectory database is then available for online querying. The state-of-the-art QDTS model, named RL4QDTS [45], employs a reinforcement learning model and takes into account the relationship between trajectories and queries during simplification.

Example 1. *Fig. 1 provides an example of RL4QDTS. Given four trajectories $T_i$ $(1 \leq i \leq 4)$ comprising a set of points $P$ $(|P| = 22)$ and a compression rate $cr = 0.5$, RL4QDTS has two major steps. First, RL4QDTS includes the start and end points of each trajectory in the simplified set $S = \{p_{1,1}, p_{2,1}, p_{3,1}, p_{4,1}, p_{1,5}, p_{2,7}, p_{3,4}, p_{4,6}\}$. Second, RL4QDTS iteratively simulates queries $Q_1$, $Q_2$, and $Q_3$ (blue rectangulars in Fig. 1). In the first iteration, it selects a region $R_1$ based on $Q_i$ $(1 \leq i \leq 3)$ and adds $p_{4,4}$ that falls within $R_1$ (grey grid cells in Fig. 1) to $S$. In the second iteration, it updates the states based on the selected region $R_1$, selects $R_2$, and adds $p_{1,4}$ to $S$. This procedure is repeated until $\frac{|S|}{|P|} = cr$, with a higher cr indicating a higher compression rate. The final simplified trajectory set is $S = \{T_1^* = \langle p_{1,1}, p_{1,4}, p_{1,5}\rangle, T_2^* = \langle p_{2,1}, p_{2,7}\rangle, T_3^* = \langle p_{3,1}, p_{3,4}\rangle, T_4^* = \langle p_{4,1}, p_{4,4}, p_{4,6}\rangle\}$.*

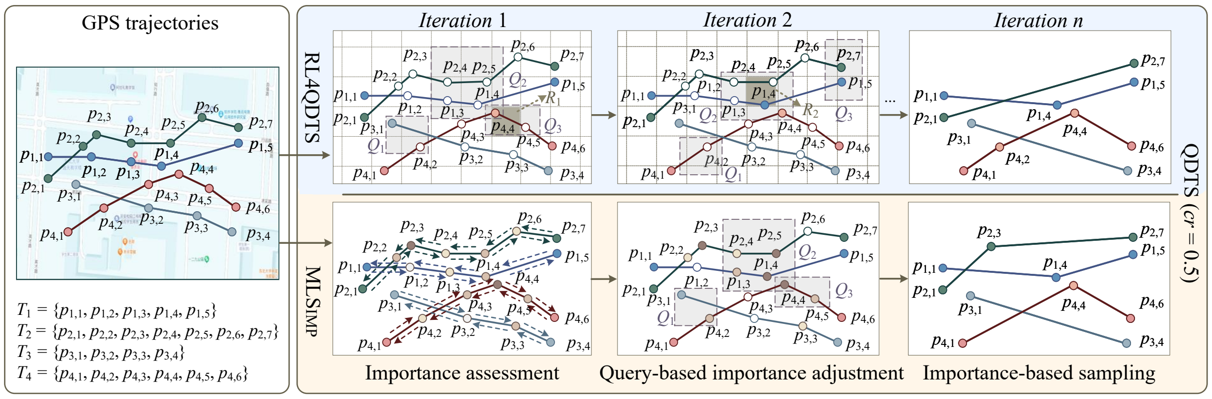

GPS trajectories
RL4QDTS
MLSimp
QDTS (cr = 0.5)
Iteration 1
Iteration 2
Iteration n
Importance assessment
Query-based importance adjustment
Importance-based sampling
$T_1 = \{p_{1,1}, p_{1,2}, p_{1,3}, p_{1,4}, p_{1,5}\}$
$T_2 = \{p_{2,1}, p_{2,2}, p_{2,3}, p_{2,4}, p_{2,5}, p_{2,6}, p_{2,7}\}$
$T_3 = \{p_{3,1}, p_{3,2}, p_{3,3}, p_{3,4}\}$
$T_4 = \{p_{4,1}, p_{4,2}, p_{4,3}, p_{4,4}, p_{4,5}, p_{4,6}\}$

**Figure 1: Circles denote GPS points, with solid lines connecting them to form trajectories. Colored circles indicate points retained in simplified trajectories, while white circles show points that are not retained by the current iteration.**

While existing studies demonstrate effectiveness in trajectory simplification, they face three major challenges:

1) ***How to compute the importance of GPS points non-iteratively?*** Trajectory simplification typically calculates the importance of GPS points iteratively, as their significance changes with updates to the simplified trajectory [7, 9, 14, 30, 44]. For example in Fig. 1, when $p_{4,4}$ is included, the importances of its neighboring points $p_{4,3}$ and $p_{4,5}$, as determined by existing EDTS methods [7, 14], are likely to decrease. Moreover, EDTS methods [7, 14, 28, 29, 31] typically rely on dynamic programming or binary search, which are time-consuming. The recent methods RLTS [44] and RL4QDTS [45], as shown in Example 1, implement an iterative process due to its use of reinforcement learning, resulting in substantial time overhead (as discussed in Sec. 5.3).

2) ***How to retain "global information" in simplified trajectories for adapting to diverse queries?*** The effectiveness of range-search queries (e.g., point, range, and window) is affected significantly by the specific distribution of individual locations, referred to as *local information* [2]. Conversely, similarity-based queries (e.g., *k*NN queries, similarity search, and clustering) depend on the overall structure of a trajectory, referred to as *global information* [2]. Existing studies [7, 14, 17, 18, 26, 27, 30–33, 35, 44, 45] typically simplify GPS points while focusing solely on local information, ignoring global information. For example, for trajectory $T_1$, an existing study [30] measures the importance of $p_{1,3}$ based on its distance to the line segment $\overline{p_{1,4}p_{1,5}}$. Recent studies [3, 8, 16, 51] employing trajectory embeddings aim to preserve similarity in the embedding space, but often fail to represent accurately the distribution space of simplified trajectories, especially when having to capture global information across extended trajectories.

3) ***How to identify the most important point for simplified trajectories from points with similar importance at low compression rates?*** Existing studies [7, 14, 30, 31] typically select either the most important point to retain or the least important point to drop. However, candidate points often have equal importance and the "most important" one is chosen randomly. At low compression rates—where few points are retained—this random selection can cause substantial deviations from the original trajectory. For example in Fig. 1, when simplifying trajectory $T_2$ [7], initially, only the start and end points are retained. Next, the importances of points $p_{2,i}$ ($2 \leq i \leq 6$) are measured by their Euclidean distance to the segment $\overline{p_{2,1}p_{2,7}}$, and $p_{2,3}$ is selected. In the next iteration, $p_{2,5}$ and $p_{2,6}$ have equal distances to $\overline{p_{2,3}p_{2,7}}$, and $p_{2,6}$ is chosen randomly. If the iteration ends here due to the low compression rate, this choice results in more information loss than if $p_{2,5}$ was selected, as $p_{2,5}$ is better for maintaining $T_2$'s shape. This highlights the need for means of more accurately discerning point importance, especially at low compression rates, ensuring the retention of the most important points.

To address the above challenges, we introduce a novel **M**utual **L**earning query-driven trajectory **simp**lification method (MLSimp). We define two key concepts to quantify point importance: *globality* and *uniqueness*. Globality captures the correlations of GPS points with their entire trajectory, while uniqueness captures the differences between a point and its neighboring points. MLSimp incorporates a lightweight Graph Neural Network (GNN)-based trajectory simplification (GNN-TS) model. For the first challenge, the GNN-TS models a trajectory as a graph with points as nodes. The GNN-TS enables parallel generation of node embeddings, avoiding iterative updates dependent on a trajectory's current state.

To address the second challenge, the GNN-TS measures point importance using both globality and uniqueness, which enables consideration of both the global and local information of a trajectory. Moreover, the GNN-TS incorporates attention mechanisms in its GNN layers. This allows points to dynamically integrate data from adjacent nodes and refine their representations, thereby preserving essential trajectory details. To address the third challenge, we introduce Diff-TS, a complex diffusion-based trajectory simplification model, and integrate it with the GNN-TS in the MLSimp mutual learning framework. In this framework, the GNN-TS provides simplified trajectories with high compression rates as soft labels for Diff-TS, which, in turn, offers low compression rate trajectories as feedback to enhance GNN-TS training. This facilitates a clearer distinction between the importance of GPS points, based on data inferred from the integrated model. Subsequently, MLSimp adjusts point importance based on simulated queries and generates simplified trajectories by performing sampling.

Example 2. *Fig. 1 illustrates MLSimp applied to trajectories $T_i$ ($1 \leq i \leq 4$). The dashed arrows indicate the direction of information passing. Each point $p_i$ ($1 \leq i \leq 22$) aggregates global and local information from other points in the same trajectory, allowing for simultaneous information exchange and thus avoiding iterations. In trajectory*

*$T_2$, although $p_{2,5}$ and $p_{2,6}$ initially have similar importance due to comparable semantics, the Diff-TS model subsequently decreases $p_{2,6}$ importance and increases that of $p_{2,5}$ based on simplified point generation conditioned on $T_2$. Next, we adjust the importance based on the distribution of three generated queries $Q_i$ ($1 \leq i \leq 3$). The importances of $p_{1,3}$, $p_{1,4}$, $p_{2,4}$, $p_{2,5}$, $p_{4,2}$, $p_{4,4}$, and $p_{4,5}$ are elevated as they are included in range queries, whereas the remaining points have their importance reduced as they are not in the range queries. Finally, we sample three GPS points based on the importance and obtain the final simplified trajectory dataset $S = \{T_1^* = \langle p_{1,1}, p_{1,4}, p_{1,5}\rangle, T_2^* = \langle p_{2,1}, p_{2,3}, p_{2,7}\rangle, T_3^* = \langle p_{3,1}, p_{3,4}\rangle, T_4^* = \langle p_{4,1}, p_{4,4}, p_{4,6}\rangle\}$.*

Our contributions are summarized as follows:

- We introduce MLSimp, a novel mutual learning framework for QDTS. It alternates training between two models, GNN-TS and Diff-TS, but employs only the GNN-TS for simplification. To the best of our knowledge, this is the first mutual learning framework and the first application of GNNs and diffusion models in trajectory simplification.
- The GNN-TS model evaluates the importance of GPS points using two new metrics: globality and uniqueness. It aggregates data from all points in the same trajectory simultaneously to eliminate iterations and captures global information efficiently.
- The Diff-TS model processes simplified trajectories with high compression rates from the GNN-TS to train its diffusion model. This feedback, in the form of low compression trajectories, sharpens the distinction in the importance of GPS points through insights from the combined model.
- We report on experiments with eight state-of-the-art methods on three datasets. The results show that MLSimp not only can reduce the simplification time by 44%–70% but also enhances query accuracy by up to 34.6%.

We review related work in Section 2 and cover preliminaries in Section 3. Section 4 presents MLSimp, while Section 5 discusses experimental findings. Section 6 concludes the paper and outlines research directions.

## 2 RELATED WORK

Trajectory simplification can be performed in two modes: online and batch. In online mode [9, 24, 32, 33, 35, 44, 48], sensors continuously collect trajectory data, storing it temporarily in a local buffer. This mode aims to select key trajectory points to be saved on the server. This mode is suitable in scenarios that require real-time updates, such as vehicle tracking, live sports analytics, and dynamic route optimization. Conversely, in batch mode [7, 14, 30, 31, 45], all trajectory data is pre-stored in the database, and there are no updates. This mode reduces simplification errors and optimizes storage space, making it suitable in scenarios requiring high precision and data usability, such as historical data analyses and offline route optimization. We review the studies on the batch mode, which is the focus of this paper.

### 2.1 Error-Driven Trajectory Simplification

*2.1.1 Non-learning-based methods.* Many methods simplify trajectories using a top-down approach [7, 14, 31]. The DP method [7] simplifies trajectories by recursively splitting them based on an error threshold, using the point with the maximum perpendicular Euclidean distance to dictate splits. To improve efficiency, DPhull [14] leverages convex hull properties to identify significant points, reducing the computational complexity while maintaining the same output. Another extension of DP, TD-TR [31], incorporates time, using synchronized Euclidean distance to measure errors.

In contrast, Bottom-Up trajectory simplification [30] begins with individual GPS points and gradually aggregates them into segments until a specified error threshold is satisfied. This method focuses on gradually reducing errors by merging segments while preserving critical trajectory characteristics. Despite its effectiveness at preserving location information, the method may omit critical points that are important for clustering and querying. To address this, the DPTS method [28] retains both directional and positional information by considering angular distances.

*2.1.2 Learning-based methods.* RLTS [44] uses reinforcement learning for trajectory simplification. It models trajectory simplification as a Markov decision process and employs policy gradient methods to learn simplification policies. Although it achieves minimal error, the high time cost of training poses challenges. To address this, S3 [9] leverages two Seq2Seq models that utilize BiRNNs [36] to compress and then reconstruct trajectories, reducing both training and simplification times. Despite higher error rates compared to RLTS, S3 lowers the simplification time considerably. EB-OTS [46] also employs reinforcement learning for trajectory simplification, targeting a minimal compression rate given a fixed error threshold. In contrast, MLSimp assumes a fixed compression rate and then performs simplification to meet that rate (see Example 1). Thus, MLSimp is not compared against EB-OTS.

In summary, while the majority of trajectory simplification studies [7, 14, 30, 31, 44] seeks to minimize error, they often overlook enhancing query correctness—a crucial simplification goal in real-world applications. In Section 5, we include DPHull [14], Bottom-Up [30], RLTS [44], and S3 [9] which are considered as baselines.

### 2.2 Query-Driven Trajectory Simplification

Error-driven trajectory simplification methods treat trajectory compression and storage as independent components. However, in a larger trajectory management system, trajectories are often stored and indexed to support query processing or pattern mining. Thus, Zhang et al. [49] propose considering data availability as a simplification quality measure. Inspired by this work, Wang et al. [45] introduce the Query-Driven Trajectory Simplification (QDTS) problem. They aim to find a simplified trajectory database within a given storage budget while preserving query accuracy as much as possible on the simplified database. They present a reinforcement learning-based method, RL4QDTS. This method works in a top-down manner. RL4QDTS establishes an octree index based on the trajectory database, defining the selection process as two decision tasks. The model learns the selection strategy from the difference between the query results of the original and the simplified database for a set of range queries. The simplified database not only supports range queries but also effectively supports $k$NN queries, similarity queries, and clustering.

However, RL4QDTS still has shortcomings: First, although it works in a top-down manner, it remains an iterative simplification method that requires multiple steps. Second, RL4QDTS no longer considers whether the simplified trajectory still retains the key information of the original trajectory. It relies solely on training the model based on range queries, making it easy to lose critical information during simplification, thereby reducing the accuracy of similarity-based queries (such as $k$NN queries). In contrast, MLSimp mines the semantics of trajectories, predicts the importance of each GPS point based on the information contained in each point, and then adjusts the importance based on the generated range query workload. MLSimp considers trajectory and query information simultaneously while avoiding iterations.

## 3 PRELIMINARIES

Definition 1. *A GPS **point** $(x, y, t)$ records the longitude $x$ and the latitude $y$ at time $t$.*

Definition 2. *A **trajectory** $T$ is a sequence of GPS points, i.e., $T = \langle p_1, p_2, \ldots, p_n \rangle$, where $p_i.t < p_{i+1}.t (1 \le i \le n-1)$.*

Definition 3. *Given a database $D$ of trajectories, a **range query** $Q_{range} = (x_{min}, x_{max}, y_{min}, y_{max}, t_{min}, t_{max})$ finds all trajectories that contain at least one point $p_i = (x_i, y_i, t_i)$ such that $x_{min} \le x_i \le x_{max}$, $ymin \le y_i \le y_{max}$, and $t_{min} \le t_i \le t_{max}$.*

Definition 4. *Given a trajectory database $D$, a **kNN query** $Q_{kNN} = (k, T_q, [t_s, t_e])$ finds a set $R$ of $k$ trajectories such that $\forall T_i \in R; \forall T_j \in D - R; (\Theta(T_q[t_s, t_e], T_i[t_s, t_e]) \le \Theta(T_q[t_s, t_e], T_j[t_s, t_e]))$, where $\Theta(\cdot, \cdot)$ represents a dissimilarity measure for trajectories.*

In this paper, we use EDR [5] to instantiate $\Theta(\cdot, \cdot)$. However, our proposals are orthogonal to the dissimilarity measure used.

Definition 5. *Given a trajectory database $D$, a **similarity query** $Q_{sim} = (T_q, [t_s, t_e], \Delta)$ finds a set $R$ of trajectories defined as $R = \{T \in D | \forall i \in [t_s, t_e] (d(T_q[i], T[i]) \le \Delta)\}$ where $T_j \in R$, $d(\cdot, \cdot)$ denotes Euclidean distance.*

Definition 6. *Given a trajectory database, trajectory **clustering** [22] partitions each trajectory into subtrajectories and then clusters subtrajectories based on some notion of trajectory distance.*

Definition 7. ***Trajectory simplification** aims to eliminate points from a trajectory $T$ to obtain a simplified trajectory $T'$ of the form $T' = \langle p_{s_1}, p_{s_2}, \ldots, p_{s_m} \rangle$,*

Definition 8. *Given a trajectory database $D$ and a storage budget $W$ indicating a fraction $r$ of the original points in $D$ to be retained, **Query-Driven Trajectory Simplification** [45] aims to find a trajectory database $D'$ of simplified trajectories, such that the difference between query results on $D$ and $D'$ are minimized.*

## 4 MUTUAL LEARNING TRAJECTORY SIMPLIFICATION

### 4.1 GNN-based Trajectory Simplification Model

*4.1.1 Framework.* GNN-TS consists of three steps: trajectory encoding, trajectory graph construction, and GNN importance prediction, as shown in Fig. 2.

**Trajectory Encoding**. Given a trajectory $T$, a pre-trained Trajectory Bert (T-Bert) encodes each point, generating embeddings for points. We detail T-Bert in Sec. 4.1.2.

**Trajectory Graph Construction**. Trajectory segment embeddings are generated from the embeddings of GPS points. A long trajectory is constructed into a graph $G_T$ using trajectory segments and points, as described in Sec. 4.1.3.

**GNN Importance Prediction**. The GNN aggregates hidden information from multiple segments to update node embeddings by $G_T$. It analyzes the uniqueness and globality of embeddings to predict their importance and construct a self-supervised contrastive learning loss. Importance is combined with amplification signals provided by Diff-TS to form a mutual learning loss (ML loss). Both contrastive and mutual learning losses are used to train the GNN. The prediction and training process is detailed in Sec. 4.1.4.

*4.1.2 Trajectory Encoding.* Given a trajectory $T = \langle p_1, p_2, \ldots, p_n \rangle$, we partition it into trajectory segments, each containing $w$ points. The trajectory segment sequence is denoted as $S_T = \langle seg_1, seg_2, \ldots, seg_m \rangle$, where $seg_k = \langle p_k^j, p_k^{j+1}, \ldots, p_k^{j+w} \rangle$ $(1 \le k \le m)$. Each segment is encoded separately. If the last segment, $seg_m$, has fewer than $w$ points, it is automatically padded to $w$ points during processing. Next, we detail the encoding of the points in a segment.

T-Bert first encodes the location and time of point $p_i = (x_i, y_i, t_i)$, where $p_i \in seg_k$ and $seg_k \in S_T$, as inputs to the Transformer layers through a spatiotemporal encoder. For location encoding, we employ node2vec [11] to capture the location information of GPS points. To capture temporal differences of points, we follow an existing study [47] and use a set of trainable parameters to encode time. Thus, the initial encoding of $p_i$ is $\mathbf{z}_i = \mathbf{z}_i^t + \mathbf{z}_i^l$, where $\mathbf{z}_i^t$ and $\mathbf{z}_i^l$ denote the temporal and spatial embeddings of point $i$, respectively.

The resulting sequence of vectors of each segment is then fed into a stacked Transformer encoder layer [39], where each layer consists of a multi-head self-attention layer and a feed-forward neural network, generating embedding for each point based on its trajectory segment. The $l$-th layer output embedding for $p_i$ of the Transformer is denoted as:

$$\mathbf{h}_i^{(l)} = \texttt{FFN}(\texttt{MultiHeadAttention}(\boldsymbol{h}_i^{(l-1)})), \tag{1}$$

where $\texttt{MultiHeadAttention}(\cdot)$ is the multi-head self-attention layer, $\texttt{FFN}(\cdot)$ is the feed-forward neural network and $\mathbf{h}_i^{(0)} = \mathbf{z}_i$.

Inspired by an existing study [6], we employ a Masked Language Model (MLM) to construct a self-supervised training task. Given a trajectory $T$, we randomly mask 20% of the points using a special token $[mask]$. For a masking set $M$, we train the model by predicting the value of its original token.

*4.1.3 Trajectory Graph Construction.* Using T-Bert, we obtain the output of the last layer of the Transformer as the embedding for each point, i.e., $\boldsymbol{h}_i = \boldsymbol{h}_i^{(L)}$. For each segment $seg_i = \langle p_i^1, ..., p_i^w \rangle$, we compute its embedding $\boldsymbol{h}_{seg_i}$ through average pooling.

To explore the relationships between GPS points and other segments in trajectory $T$, we construct a trajectory graph.

Definition 9. *The **trajectory graph** $G_T = (V, E)$ of trajectory $T$ is an undirected graph, where $V$ and $E$ are the set of nodes and edges, respectively. The node set $V = V_p \cup V_{seg}$, where $V_p = \{v_1, v_2, \ldots, v_n\}$ is the set of all GPS points in $T$, and $V_{seg} = \{v_{seg_1}, v_{seg_2}, \ldots, v_{seg_m}\}$ is the segments of $T$. Each segment node is connected to each GPS node via an edge in $E$. The feature matrix $\boldsymbol{H} = [\boldsymbol{H}_p, \boldsymbol{H}_{seg}]$ that contains the initial representation of the GPS point and segment nodes: $\boldsymbol{H}_p = [\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_n]$ and $\boldsymbol{H}_{seg} = [\boldsymbol{h}_{seg_1}, \boldsymbol{h}_{seg_2}, \ldots, \boldsymbol{h}_{seg_m}]$.*

The trajectory graph $G_T$ in Fig. 2 has $V_p = \{v_1, v_2, v_3, \ldots, v_7\}$, $V_{seg} = \{v_{seg_1}, v_{seg_2}, v_{seg_3}\}$, and $E = \{(v_i, v_{seg_j}) | 1 \le i \le 7, 1 \le j \le 3\}$. Next, we define a feature vectors of the point $p_i$ and segment $v_{seg_j}$ node are $\boldsymbol{h}_i$ and $\boldsymbol{h}_{seg_j}$, respectively.

The trajectory partitioning and graph construction methods we propose are designed to address the input limitations of the Transformer encoder. This is inspired by the encoding of long

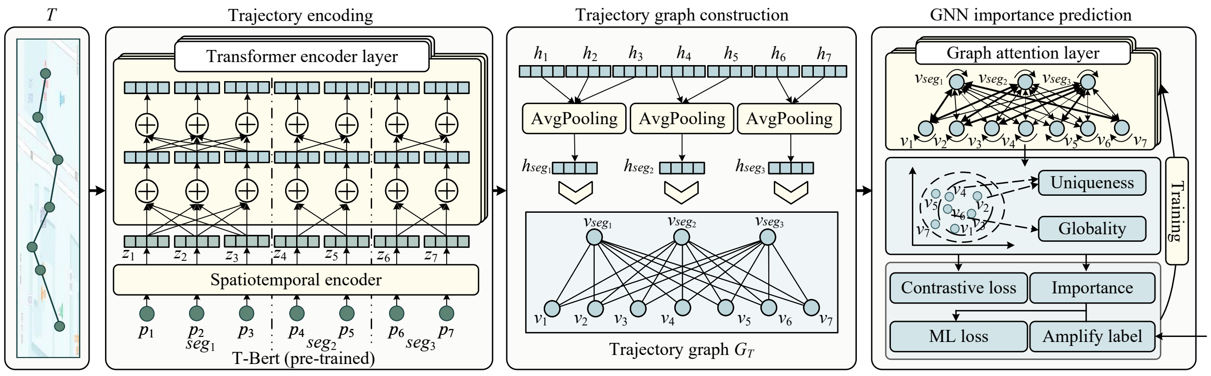


**Figure 2: GNN-TS model overview.**

text, as discussed elsewhere [1, 12]. Unlike existing techniques that involve trajectory partitioning, we do not require goal-oriented segmentation. For example, the goal of one study [37] is to consider the homogeneity in the neighborhoods of space-time points, and the goal of another study [19] is to minimize the cost of multicut.

*4.1.4 GNN Importance Prediction.* Using the trajectory graph $G_T$, we update the representation of trajectory nodes using a GNN. GNNs [15, 20, 21, 38, 40] are neural networks that operate on graph-structured data, allowing for efficient aggregation of node information from edges. Here, we use a GNN consisting of Graph Attention Network (GAT) [40] layers. GATs can learn complex relationships between nodes, and their attention mechanism allows the model to focus on specific nodes in a graph. Specifically, given a node $i$ and its neighbor node set $\mathcal{N}_i$, we propagate the features as:

$$\boldsymbol{g}_i^{(l)} = \texttt{GATLayer}(\boldsymbol{g}_i^{(l-1)}, \boldsymbol{g}_{\mathcal{N}_i}^{(l-1)}), \tag{2}$$

where $\texttt{GATLayer}(\cdot)$ is the propagation function of a GAT layer, $1 \le l \le L$, $L$ is the number of GAT layers, and $\boldsymbol{g}_i^{(0)} = \boldsymbol{h}_i$.

Using the $L$ layers of GAT, the node representation $\boldsymbol{g}_i = \boldsymbol{g}_i^{(L)}$ learns the representations of GPS points across the entire trajectory rather than just containing information within the current segment. To quantify the importance of each point, we analyze the distribution of point representations and define two metrics for trajectory importance: uniqueness and globality.

**Uniqueness**. In a trajectory, due to the short sampling interval (usually only a few seconds), neighboring points are often highly similar, leading to redundancy, as illustrated by points $p_{2,2}$ and $p_{2,3}$ of $T_2$ in Fig. 1. To reduce this redundancy, we measure the uniqueness of points relative to the embeddings of neighboring points. In the uniqueness calculation, neighbors consist of the $k$ nearest trajectory points to $p_i$, where $k$ is a predefined parameter. These points are selected based on their cosine similarity to $p_i$'s representation vector; the higher the similarity, the closer the point to the target. This selection process ensures that the chosen neighbors are semantically similar to $p_i$, effectively filtering out points that significantly differ during the comparison. The uniqueness of a point $p_i$ represented by $\boldsymbol{g}_i^{(L)}$ is calculated as follows:

$$\mathcal{L}_{uni}(\boldsymbol{g}_i) = \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \|\boldsymbol{g}_i - \boldsymbol{g}_j\|_2, \tag{3}$$

where $\mathcal{N}_i$ is the neighbors of point $p_i$ in $T$.

**Globality**. The goal of trajectory simplification is to select points that best represent the original trajectory. Therefore, points that capture the semantics of the trajectory are important, such as the starting point $p_{4,1}$, end point $p_{4,6}$, and turning point $p_{4,4}$ of $T_4$ in Fig. 1. Hence, we define globality to capture the similarity of a point with all other points in its trajectory, using the representations of the points. If a point's globality is high, the point is important. Conversely, if the similarity between a point and all other points is low, the GPS point may be noisy or simply wrong. Globality is defined as follows:

$$\mathcal{L}_{glob}(\boldsymbol{g}_i) = \texttt{log}\left(\frac{1}{|T|-1} \sum_{j=1, j\neq i}^{|T|} e^{-2\|\boldsymbol{g}_i - \boldsymbol{g}_j\|_2^2}\right), \tag{4}$$

where $|T|$ is the total number of points in trajectory $T$.

To better capture both semantic similarities and relationships between points, we align uniqueness and globality using contrastive learning. Contrastive learning [43] is a self-supervised training method based on the alignment and uniformity of the distribution of all objects.

During training, we treat the neighboring points of a point $p_i$ as its positive examples to encourage similarity in the embedding space (i.e., minimizing $\mathcal{L}_{uni}$). Conversely, we treat distant points from $p_i$ as negative examples to ensure separation (i.e. minimizing $\mathcal{L}_{glob}$). Therefore, the loss function is:

$$\mathcal{L}_{con}(\boldsymbol{g}_i) = \mathcal{L}_{uni}(\boldsymbol{g}_i) + \lambda_1 \mathcal{L}_{glob}(\boldsymbol{g}_i), \tag{5}$$

where $\lambda_1$ is a hyperparameter balancing the two terms. We fix it at 0.5 following the literature [43].

When measuring the importance of point $p_i$, we consider uniqueness and globality simultaneously and define the importance:

$$I_{p_i} = \mathcal{L}_{uni}(\boldsymbol{g}_i) \times \mathcal{L}_{glob}(\boldsymbol{g}_i) + \epsilon, \tag{6}$$

where $\epsilon$ eliminates any zero values in the importance scores. We assess the importance of each point by evaluating its uniqueness and globality. Points with high uniqueness are valued more due to their distinct semantic characteristics; while those with low uniqueness, which contribute less to the global distinction and local representativeness of the trajectory, are prioritized for removal.

We employ a mutual learning algorithm for training, where the results of the Diff-TS are used as amplified labels compared to the importance generated by GNN-TS, serving as the ML loss. This is covered in Sec. 4.3.

**Discussion.** Since long trajectories are common in databases (see Table 1), the space complexity of encoding an entire trajectory using a Transformer encoder is very costly and makes encoding inefficient. For a trajectory of length $|T|$ encoded into $d$ dimensions, the time complexity of a single-layer Transformer encoder is

$O(|T|^2 d + |T|d^2)$, and the space complexity is $O(|T|^2 + |T|d)$. Therefore, GNN-TS first segments long trajectories for encoding. Long trajectories are typically segmented into chunks of length $w$ based on the available computational capabilities. For a trajectory of length $|T|$, the number of segments is $k = \lceil |T|/w \rceil$. If all segments are encoded serially, their time complexity is $O(kw^2 d + kwd^2)$. Due to serial encoding, the space complexity is $O(w^2 + wd) < O(|T|^2 + |T|d)$. Next, we construct a trajectory graph where the number of nodes is $|T| + k$ and the number of edges is $k|T|$. Finally, we use a GAT to fine-tune the encoding of nodes , and predict the importance of trajectory points. Since the trajectory graph is sparse, with each trajectory node connected only to $k$ segment nodes, the encoding time complexity is approximately $O((|T| + k)d^2 + k|T|d)$. Therefore, the time complexity for GNN-TS to encode a trajectory is $O(kw^2 d + kwd^2) + O((|T| + k)d^2 + k|T|d) = O((kw^2 + k|T|)d + (kw + T + k)d^2)$. When $\frac{w^2}{w-1} + \frac{w+1}{w-1}d < |T|$, the time complexity of GNN-TS is lower than when using only a Transformer encoder to generate a trajectory embedding. Thus, for long trajectories (e.g., the average trajectory length exceeds 1000), segmenting with a reasonable $w$ can improve encoding efficiency markedly. However, although GNN-TS connects trajectory nodes from different segments through the trajectory graph, segmentation encoding introduces information loss. To limit this loss, $w$ in GNN-TS is generally not very small (e.g., 500 in this paper). Thus, compared to a trajectory graph containing only trajectory points with $|T|$ nodes, the number of additional segment nodes $\lceil |T|/w \rceil \ll |T|$ does not impact the GAT encoding efficiency substantially.

## 4.2 Diffusion-based Trajectory Simplification Model

*4.2.1 Training Process.* During training, since we lack ground truth optimal simplified trajectories, we instead employ a high compression rate $cr_{high}$ simplified trajectories $T*$, sampled based on the importance generated by GNN-TS. We set $0.5 \leq cr_{high} < 1$ and assume that $T^*$ retains a considerable amount of trajectory information and can serve as soft labels for training. We concatenate the original trajectory $T$ with the simplified trajectory $T^*$ and then encode the trajectory to generate trajectory embeddings. Next, the trajectory is fed into a diffusion module. Finally, the model parameters are updated using a joint diffusion and similarity loss.

**Trajectory Encoding**. We use an Transformer to encode the concatenated trajectory $T_{concat}$ mapping the points into initial representation vectors $\boldsymbol{H}_{T_{concat}} = [\texttt{concat}(\boldsymbol{H}_T, \boldsymbol{H}_{T^*})]$, where $\boldsymbol{H}_T = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n]$, $\boldsymbol{H}_{T^*} = [\boldsymbol{h}_1^*, \ldots, \boldsymbol{h}_z^*]$, $n = |T|$, and $z = |T^*|$.

**Diffusion Module**. After obtaining the input encoding $\boldsymbol{H}_{T_{concat}}$, the continuous diffusion model conditionally generates embeddings for the simplified trajectory. The diffusion model consists of a forward and a reverse process.

(i) *Forward process*: In each forward step $q(\boldsymbol{H}_\gamma | (\boldsymbol{H}_{\gamma-1})$, we gradually inject Gaussian noise $\epsilon \sim N(0, I)$ into the hidden state $\boldsymbol{H}_{\gamma-1}$ from the previous step to obtain $\boldsymbol{H}_\gamma$. Inspired by an existing study [10], we only apply noise to $\boldsymbol{H}_{T^*}$, allowing for conditional modeling by the diffusion model. After $\gamma$ forward steps, a noisy representation $\boldsymbol{H}_\gamma$ is obtained:

$$\boldsymbol{H}_\gamma = [\texttt{concat}(\boldsymbol{H}_T, \boldsymbol{H}_{T^*}^\gamma)], \quad \boldsymbol{H}_{T^*}^\gamma = N(\sqrt{1-\beta_\gamma}\boldsymbol{H}_{T^*}^{\gamma-1}, \beta_\gamma \boldsymbol{I}), \quad (7)$$

where $\gamma \in \{1, 2, ..., \Gamma\}$, $\Gamma$ is the total number of diffusion steps, $\boldsymbol{H}_{T^*}^0 = N(\boldsymbol{H}_{T^*}, \beta_0 \boldsymbol{I})$, and $\beta_\gamma \in \{\beta_0, \beta_1, \beta 2, \ldots, \beta_\Gamma\}$, where $\beta_0 < \beta_1 < \beta 2 < \ldots < \beta_\Gamma$, is a hyperparameter.

(ii) *Reverse process*: During the training, the goal of the reverse process is to recover the original $\boldsymbol{H}_{T^*}$ through denoising. Once the noisy representation of the simplified trajectory part is obtained, we perform the reverse process to remove the noise conditioned on the previous step's trajectory representation:

$$p_\theta(\boldsymbol{H}_{T^*}^{\gamma-1} | \boldsymbol{H}_{T^*}^\gamma) = N(\boldsymbol{H}_{T^*}^{\gamma-1}; \mu_\theta(\boldsymbol{H}_{T^*}^\gamma), \sigma_\theta^2(\gamma)\boldsymbol{I}), \quad (8)$$

where $\mu_\theta(\cdot)$ and $\sigma_\theta^2(\cdot)$ are models for predicting the mean and standard deviation during the forward process $q((\boldsymbol{H}_{\gamma-1}|\boldsymbol{H}_\gamma)$. Here we use stacked Transformer layers as the noise prediction model.

**Training Loss**. After $\Gamma$ rounds of the backward process, the recovered representation vectors of the simplified trajectory are $\hat{\boldsymbol{H}}_{T^*} = [\hat{\boldsymbol{h}}_1^*, \ldots, \hat{\boldsymbol{h}}_z^*]$. We train the diffusion model using the diffusion loss $\mathcal{L}_{diff}$ in an existing study [10].

We also introduce a diversity loss to reduce the similarity among generated simplified points, thus reducing redundant information in generated trajectories. We define the diversity loss as:

$$\mathcal{L}_{div}(\hat{\boldsymbol{h}}_i) = \log\left(\frac{1}{|T^*| - 1} \sum_{j=1, j\neq i}^{n} e^{-2\|\hat{\boldsymbol{h}}_i - \hat{\boldsymbol{h}}_j\|_2^2}\right), \quad (9)$$

where $|T^*|$ is the length of the simplified trajectory. Finally, we combine the diffusion loss and the diversity loss. The training loss function is:

$$\mathcal{L}_{Diff-TS} = \mathcal{L}_{diff} + \lambda_2 \mathcal{L}_{div}, \quad (10)$$

where $\lambda_2$ is a hyperparameter that balances the two losses.

*4.2.2 Inference Process.* For inference, Diff-TS first obtains the representation $\boldsymbol{H}_T$ of the original trajectory $T$, and then adds Gaussian noise to obtain the initial simplified trajectory embedding through a Markov transition. Next, we randomly sample $\alpha$ ($\alpha \ll n$) points of Gaussian noise embeddings as the initial simplified trajectory embeddings input $\boldsymbol{H}_{T'}^0 \in \mathbb{R}^{m \times d}$, where $d$ is the dimensionality of the embeddings. The concatenated input to the diffusion model is $\boldsymbol{H}_0^\# = [\texttt{concat}(\boldsymbol{H}_T^0, \boldsymbol{H}_{T'}^0)]$. The diffusion model utilizes the learned reverse denoising process (generation process) to iteratively remove Gaussian noise. After $\Gamma$ steps of diffusion, we obtain the output simplified trajectory representation $\tilde{\boldsymbol{H}}_{T'} = [\tilde{\boldsymbol{h}}_{s_1}, \tilde{\boldsymbol{h}}_{s_2}, \ldots, \tilde{\boldsymbol{h}}_{s_\alpha}]$. Then, Diff-TS computes the matching between the generated summary representation $\tilde{\boldsymbol{h}}_i$ and the original trajectory $\boldsymbol{H}_T$ representation, selects the point with the highest score, and adds it to the simplified trajectory. Finally, we obtain the simplified trajectory.

## 4.3 Mutual Learning and Simplification

GNN-TS can be viewed as an *extractive* simplification model. Similar to extractive summarization in NLP, GNN-TS selects the $l_s$ most important points as the simplified trajectory by evaluating the importance of all points. However, extractive simplification may retain redundant points, as the importance is fixed and is unaffected by the importance of already retained points. Diff-TS can be considered a *generative* simplification model. It generates a simplified trajectory by learning from the original one. However, (i) it cannot guarantee that generated points exist in the original trajectory, and (ii) the generated trajectory may deviate considerably from the original in terms of information—it may contain noise and may fail to maintain the original shape.

To address the shortcomings of GNN-TS, we use ML to integrate the two models for trajectory simplification. The unsupervised GNN-TS provides training labels for the supervised Diff-TS, ensuring that the simplified trajectories generated by Diff-TS do not deviate substantially from the original trajectories. The trajectories generated by Diff-TS are regarded as amplified signals that prompt GNN-TS to retain important points that might be lost due to redundancy. Next, we present the framework for mutual learning between the two models and the process of simplification using the trained models.

*4.3.1 Mutual Learning Training.* We iteratively train two simplification models. This is inspired by an existing sudy [50], where multiple networks learn from each other in a supervised setting. We train two models with two stages.

In the first stage, we train the GNN-TS using only the contrastive loss in a self-supervised manner, without using the mutual learning loss. The trained GNN-TS is used to infer a high compression rate simplified database. In the second stage, we perform mutual learning for both models. We use the simplified database as the labels for Diff-TS training. After training, Diff-TS infers the $\alpha$ most important points from trajectory $T$ to add these to the simplified trajectory $T^{\#}$, generating amplified labels $Y_T = \{y_1, ..., y_n\}$. If $p_i \in T^{\#}$, where $p_i \in T$, then $y_i = 1$; otherwise $y_i = 0$. We retrain the GNN-TS, introducing an ML loss:

$$\mathcal{L}_{ml} = \sum_{i=1}^{n} -y_i \log I_{p_i} - (1 - y_i)\log(1 - I_{p_i}), \tag{11}$$

where $I_{p_i}$ is the importance predicted by GNN-TS. The overall loss function of GNN-TS during the mutual learning stage is as follows:

$$\mathcal{L}_{GNN-TS} = \mathcal{L}_{con} + \lambda_3 \mathcal{L}_{ml}, \tag{12}$$

where $\lambda_3$ is a hyperparameter for balancing the two losses.

*4.3.2 Simplification.* Upon training, we use the lightweight GNN-TS for inference. Given a dataset $D$ to be simplified, we first use GNN-TS to predict the importance $I_{p_i}$ of each point $p_i \in D$ and perform global normalization. However, the effectiveness of the simplified database on range queries is not ideal, as shown in Fig. 11. Therefore, to support range queries better, we include range query-based importance adjustment.

**Importance Adjustment:** Since previous queries are not available, we synthesize a workload of range queries $Q^w$, where each query location is randomly sampled by following some distribution (e.g., a data distribution) as in the literature [45]. Through the queries, we obtain a set of query results. We divide the spatial and temporal ranges into coarse-grained grid cells, where the cells hit by the query results are assigned an importance of 1, and the remaining cells are assigned 0. Finally, we normalize the query-based importance of each cell. The query-based importance $I^q_{p_i}$ of point $p_i$ in $D$ is consistent with the importance of the cell it belongs to. The adjusted importance of point $p_i$ is defined as follows:

$$I^{adj}_{p_i} = (1 - \delta) I_{p_i} + \delta I^q_{p_i}, \tag{13}$$

where $\delta$ is the adjustment ratio based on queries.

Finally, we directly sample $m$ points with the adjusted importance to construct the simplified trajectory database, with compression rate $cr = m/n$.

*4.3.3 Complexity.* Following existing learning-based trajectory simplification models [9, 44, 45], we focus on analyzing the time complexity of simplification inference and exclude the training.

**Table 1: Trajectory dataset statistics.**

| Statistic | Geolife | T-Drive | OSM |
|---|---|---|---|
| # of trajectories | 17,621 | 10,359 | 513,380 |
| Total # of points | 24,876,978 | 17,740,902 | 2,913,478,785 |
| Ave. # of pts per traj | 1,433 | 1,713 | 5,675 |
| Sampling rate | 1s–5s | 177s | 53.5s |
| Average length | 9.96m | 623m | 180m |

For a database with $m$ trajectories, where $w$ is the maximum input length of T-Bert and $|D'|$ is the storage budget of the simplified database, the simplification process includes the following steps: (i) ***Trajectory encoding*** segments a trajectory, resulting in $\overline{n}/w$ encodings for a trajectory of length $\overline{n}$, with time complexity $O(\overline{n}/w)$. (ii) ***Trajectory graph construction*** is a logical process where, in practice, it is not necessary to construct the actual graph structure. Thus, the time complexity of trajectory graph construction is $O(1)$. (iii) ***GNN importance prediction*** consists of GAT's attention matrix computation and attention-based node feature aggregation, and importance calculation (uniqueness and globality computations). All steps are matrix computations on the feature matrix and the edge index matrix. Following learning-based trajectory studies [4, 8, 9, 44, 45], the time complexity of GNN importance prediction is $O(1)$. (iv) ***Importance adjustment*** of each trajectory point is calculated in parallel based on the adjustment values derived from the simulated range query results. Thus, the time complexity of importance adjustment is $O(1)$. (v) ***Sampling*** is performed after determining the importance of all trajectories. Thus, the sampling time complexity for the entire database is $O(1)$. Therefore, the overall simplification time complexity is $m \cdot O(\overline{n}/w + 1 + 1 + 1) + O(1) = O(m \cdot \overline{n}/w)$.

## 5 EXPERIMENTAL STUDY

### 5.1 Experimental Setup

*5.1.1 Datasets.* We evaluate MLSimp on three publicly available real-world datasets. Table 1 provides statistics of the datasets.

**Geolife**[1] is a GPS trajectory dataset collected in the Geolife project from 182 users over a period of more than three years.

**T-Drive**[2] contains one-week trajectories from 10,357 taxis. The dataset contains about 17 million points.

**OSM**[3] contains 500 million GPS trajectories with 3 billion points shared by the community on OpenStreetMap over 9 years.

*5.1.2 Competitors.* Since the existing algorithms for the QDTS problem only include RL4QDTS, the existing EDTS algorithm is also considered as a potential baseline. Guided by the skyline study in an existing study [45], we select the most competitive existing EDTS method as the target baseline.

**Top-Down(E, PED) and Top-Down(W, PED)** [14] start with the entire trajectory and iteratively divides it into segments until each segment meets an error threshold. In the following section, we use "TD" to represent "Top-Down."

[1]https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/

[2]https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/

[3]https://star.cs.ucr.edu/?OSM/GPS#center=43.6,-56.1&zoom=2

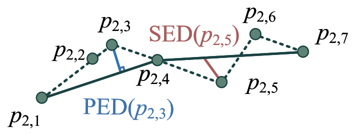


**Figure 3: An example of the SED and PED errors, where (i) circles denote trajectory points, (ii) green solid and dash lines denote the original and simplification trajectory, respectively, and (iii) the red and blue solid lines denote the SED of $p_{2,5}$ and the PED of $p_{2,3}$, respectively.**

**Bottom-Up(E, PED), Bottom-Up(E, SED), and Bottom-Up(E, DAD)** [30] start with the individual points of a trajectory and merge them iteratively to meet the compression rate. In the following section, we use "BU" to represent "Bottom-Up."

**RLTS(E, SED)** [44] employs reinforcement learning to iteratively simplify trajectories. It formulates trajectory simplification as a sequential decision-making problem, where an agent decides which points to retain or remove based on rewards associated with simplification decisions.

**S3** [9] uses two Seq2Seq models for training and generates simplified trajectories by feeding the original trajectory into an encoder-decoder consisting of Bi-LSTM units as a compressor.

**RL4QDTS** [45] performs QDTS by reinforcement learning. It considers range queries and simplifies trajectories for queries.

Top-Down(E, PED), Top-Down(W, PED, Bottom-Up(E, PED), Bottom-Up(E, SED), Bottom-Up(E, DAD), RLTS(E, SED), and S3 are EDTS baselines. RL4QDTS is a QDTS baseline. "E" and "W" denote simplifying each trajectory in the database and considering the entire database as a whole by inserting or deleting points between all points in the database, respectively.

*5.1.3 Hyperparameters and Experimental Settings.* For training, we randomly sample 1,000 trajectories and fix each $|T|$ at 1,000. The remaining trajectories are used for testing. For testing, the compression rate ($cr$) is defined as shown in Example 1. The lower the $cr$, the fewer points remain. For example, when $cr = 0.25\%$, the amount of data after simplification is reduced by 400 (i.e., 100/0.25) times compared to the original data. We follow a previous study [45] and generate range queries, $k$NN queries, similarity queries, and clustering. Queries are generated based on two distributions: the data distribution and a Gaussian distribution (with parameters $\mu = 0.5$ and $\sigma = 0.25$). The spatial window size for range queries is set to $2km \times 2km$. For all queries, the temporal window size is 7 days for Geolife, 3 hours for T-Drive, and 1 day for OSM. The similarity metric of $k$NN queries is EDR [5] with a threshold of 2 km. The value of $k$ is set to 3. The distance threshold for similarity queries is set to 5 km. For clustering, we adopt the tracking algorithm [22].

We set the hyperparameters of the baselines as suggested in the respective papers. In GNN-TS, the pre-trained T-Bert employs a transformer with 4 layers and 8 attention heads as the embedding layer, trained for 500 epochs. The embedding dimensionality is 128, and the maximum embedding sequence length is set to 500. The GNN consists of 2 layers of GATs with 4 attention heads, each producing an output with dimensionality 32. During training, the GNN-TS generates coarse-grained simplified trajectories with a compression rate of 0.5 for training the Diff-TS. The Diff-TS's encoder utilizes a transformer with 2 layers and 2 attention heads, producing output with a dimensionality of 128. The diffusion model employs a transformer with 2 layers and 2 attention heads, with an output dimensionality of 128. The diffusion steps are set to 500. Diff-TS generates amplified labels for simplified trajectories containing only 20 trajectory points. The hyperparameter $\lambda_1$ in $\mathcal{L}_{con}$ is fixed at 0.5 [43]. The hyperparameter $\lambda_2$ is used to balance the two terms in $\mathcal{L}_{Diff-TS}$. The hyperparameter $\lambda_3$, set to 0.5. The parameter for query-based adjustment $\delta$ is set to 0.5 (0.7 for T-Drive). We perform 100 range queries for the query-based importance adjustment. All experiments are conducted on a server with an Intel(R) Xeon(R) W-2155 CPU, 128GB memory, and an NVIDIA TITAN RTX GPU.

*5.1.4 Metrics.* We use the $F_1$-score to evaluate the performance of queries on the simplified database. For each query distribution, we generate 100 queries and report the average $F_1$-score. Given a query $Q \in \{Q_{range}, Q_{kNN}, Q_{sim}\}$, where $Q_{range}$ is a range query, $Q_{kNN}$ is a $k$NN query, and $Q_{sim}$ is a similarity query, and given the query results $R_s$ on the simplified database and $R_o$ on the original database, the $F_1$-score is:

$$F_1\text{-score}(Q) = \frac{2P(Q)R(Q)}{P(Q) + R(Q)}, \tag{14}$$

where $P(Q) = \frac{|R_s \cap R_o|}{|R_s|}$, $R(Q) = \frac{|R_s \cap R_o|}{|R_o|}$, and $|\cdot|$ returns the cardinality of its argument set.

For clustering queries, following the literature [49], we introduce $C$ to represent the clustering results on a database $D$. If $T_i$ and $T_j$ ($T_i, T_j \in D$) belong to the same cluster then $(T_i, T_j) \in C$. Given a clustering $Q_c$, the clustering results $C_s$ on the simplified database, and the clustering results $C_o$ on the original database, the $F_1$-score is defined as follows:

$$F_1\text{-score}(Q_c) = \frac{2P(Q_c)R(Q_c)}{P(Q_c) + R(Q_c)}, \tag{15}$$

where $P(Q_c) = \frac{|C_s \cap C_o|}{|C_s|}$ and $R(Q_c) = \frac{|C_s \cap C_o|}{|C_o|}$.

We use SED and PED to evaluate the simplification errors for simplification methods. We first define the anchor segment of each point in the original trajectory.

Definition 10. *Given an original trajectory $T = \langle p_1, p_2, \ldots, p_n \rangle$ and a simplified trajectory $T' = \langle p_{s_1}, p_{s_2}, \ldots, p_{s_m} \rangle$ ($m \ll n$ and $1 = s_1 < s_2 < \cdots < s_m = n$), the **anchor segment** of a point $p_i$ ($1 \leq i \leq n$) is $AncSeg_{p_i} = \overline{p_{s_j} p_{s_{j+1}}}$, where $\overline{p_{s_j} p_{s_{j+1}}}$ is a trajectory segment of $T'$ and $s_j \leq i \leq s_{j+1} - 1$.*

The simplification error $\epsilon(T')$ of a simplified trajectory $T'$ is defined as follows:

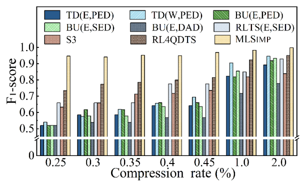
$$\epsilon(T') = \max_{1 \leq j \leq m-1} \max_{s_j \leq i < s_{j+1}} \epsilon(p_i), \tag{16}$$

where $\epsilon(p_i)$ is the error of the trajectory point $p_i$ in $T$.

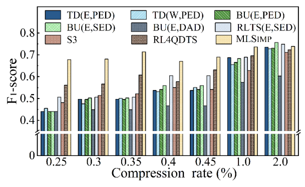
The similarity errors PED and SED for each point are shown in Fig. 3. PED is the shortest distance from $p_i$ to the anchor segment $AncSeg_{p_i} = \overline{p_{s_j} p_{s_{j+1}}}$. SED is the Euclidean distance between the trajectory point $p_i = (x_i, y_i, t_i)$ and its mapped point $p'_i = (x'_i, y'_i, t_i)$ on the anchor segment $AncSeg_{p_i} = \overline{p_{s_j} p_{s_{j+1}}}$ of $T'$ based on $t_i$:

$$\begin{aligned} x'_i &= x_{s_j} + \frac{t_i - t_{s_j}}{t_{s_{j+1}} - t_{s_j}} \cdot (x_{s_{j+1}} - x_{s_j}) \\ y'_i &= y_{s_j} + \frac{t_i - t_{s_j}}{t_{s_{j+1}} - t_{s_j}} \cdot (y_{s_{j+1}} - y_{s_j}) \\ \mathsf{SED}(p_i) &= \sqrt{(x_i - x'_i)^2 + (y_i - y'_i)^2}, \end{aligned} \tag{17}$$

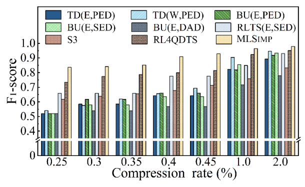
where $p_{s_j} = (x_{s_j}, y_{s_j}, t_{s_j})$ and $p_{s_{j+1}} = (x_{s_{j+1}}, y_{s_{j+1}}, t_{s_{j+1}})$.



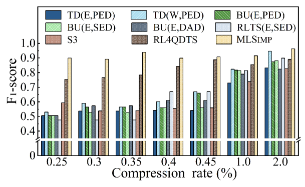
(a) **Range query**



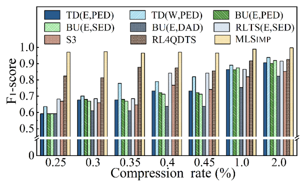
(b) ***k*NN query**



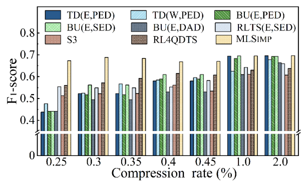
(c) **Similarity query**



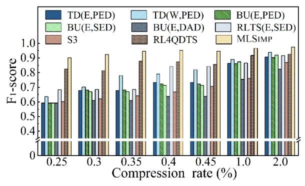
(d) **Clustering**



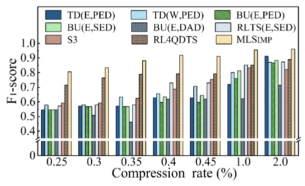
(e) **Range query**



(f) ***k*NN query**

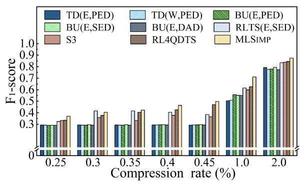


(g) **Similarity query**

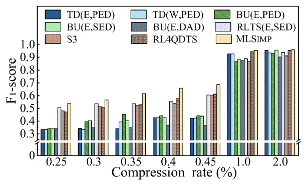


(h) **Clustering**

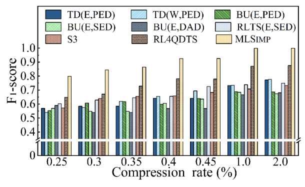
**Figure 4: $F_1$-score with different compression rate (%) on Geolife, where (a)–(d) are queries following the data distribution and (e)–(h) are queries following a Gaussian distribution.**



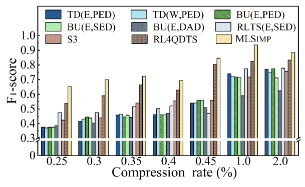
(a) **Range query**



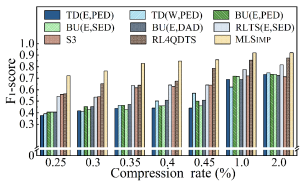
(b) ***k*NN query**



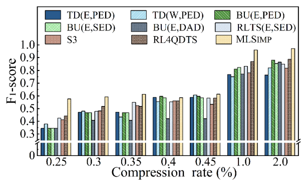
(c) **Similarity query**



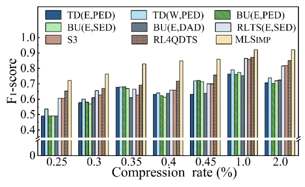
(d) **Clustering**



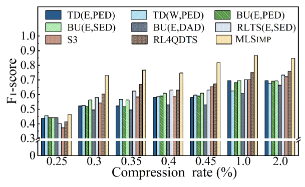
(e) **Range query**



(f) ***k*NN query**



(g) **Similarity query**



(h) **Clustering**

**Figure 5: $F_1$-score with different compression rate (%) on T-Drive, where (a)–(d) are queries following the data distribution and (e)–(h) are queries following a Gaussian distribution.**

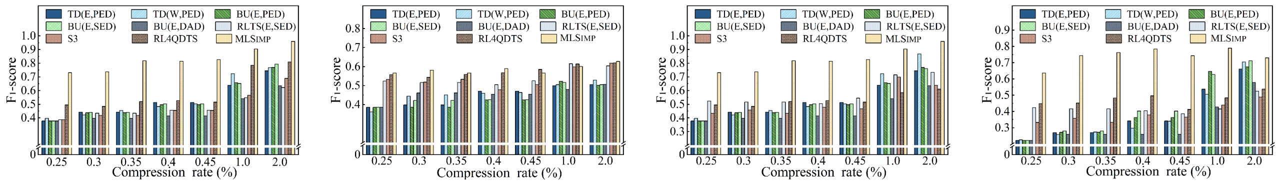
## 5.2 Effectiveness Evaluation

Figs. 4, 5, and 6 report the $F_1$-scores of all baselines on the Geolife, T-drive, and OSM datasets with 2,000,000 trajectory points, respectively. MLSimp tops consistently across all datasets and queries. Even at a fairly low compression rate of 0.25%, which means that we reduce the data by a factor of 400, most query scores exceed 0.5. Compared to the state-of-the-art method, RL4QDTS, MLSimp achieves up to a 21% improvement in range query performance, up to an 11.8% improvement in $k$NN query performance, up to a 34.6% improvement in clustering, and up to a 20.3% improvement in similarity queries on the three datasets. This is because MLSimp not only considers the impact of range queries on trajectory simplification but also, more importantly, preserves key points in the simplified database by mining global information, better fitting the trajectory similarity of the original database.

## 5.3 Efficiency Evaluation

We study the simplification efficiency of MLSimp. Fig. 7 reports the simplification time required by all methods discussed in Sec. 5.2. We report the overall simplification time rather than the time for simplifying individual trajectories. For RL4QDTS and MLSimp, we report the simplification time separately for queries generated based on the data distribution and queries generated based on a Gaussian distribution. Across the three datasets, MLSimp generates simplified results in less than 60 seconds for all compression rates and datasets. Compared to the fastest Top-Down method, the average compression time is reduced by 42%–70%.

Fig. 8 further reports the overall simplification time of MLSimp on Geolife, T-drive, and OSM, including the time for each component. The components measured include trajectory encoding, trajectory graph construction, importance prediction, importance adjustment, and trajectory sampling. In the same dataset, the time cost of trajectory encoding, trajectory graph construction, and importance prediction in GNN-TS is unaffected by varying compression rates and query distributions. This is because GNN-TS is only used to obtain global information of trajectory points and to predict importance based on this information. The main time cost during MLSimp simplification comes from the inference on the learned models T-Bert and GAT in the GNN-TS, as these models are multilayered, with each layer involving multiple matrix computations. The second highest time cost is from the importance adjustment, influenced by simulated query results. The subsequent adjustment of trajectory points varies across different query distributions. Finally, the sampling time increases with the compression rate but


(a) Range query (b) *k*NN query (c) Similarity query (d) Clustering

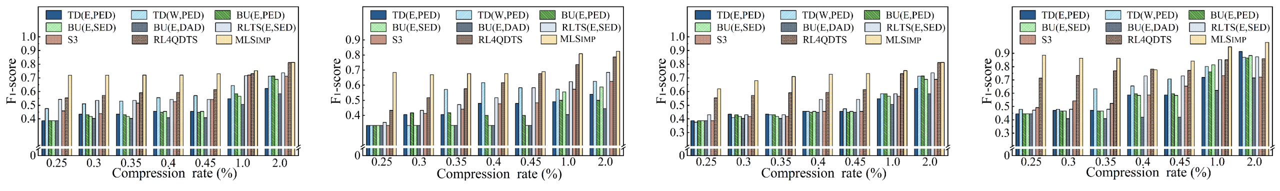

(e) Range query (f) *k*NN query (g) Similarity query (h) Clustering

**Figure 6: $F_1$-score with different compression rate (%) on OSM, where (a)–(d) are queries following the data distribution and (e)–(h) are queries following a Gaussian distribution.**

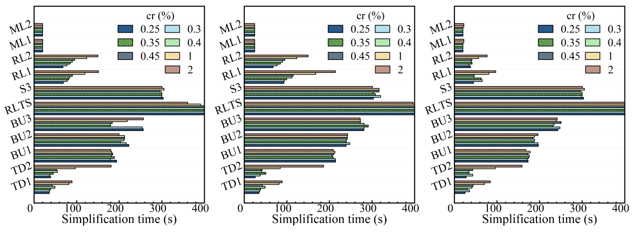
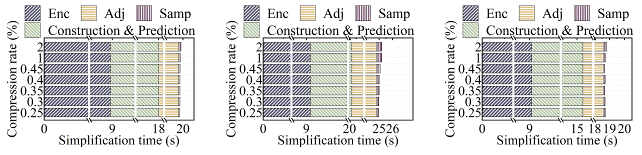

(a) Geolife (b) T-Drive (c) OSM

**Figure 7: Simplification time with different compression rates (cr), where (i) TD1 and TD2 represent TD(E, PED) and TD(W, PED); (ii) BU1, BU2, and BU3 represent BU(E, DAD), BU(E, PED), and BU(E, SED); (iii) RLTS represents RLTS(E, SED); (iv) RL1 and RL2 represent RL4QDTS for queries generated based on the data and Gaussian distribution, respectively; (v) ML1 and ML2 represent MLSimp for queries generated based on the data and Gaussian distribution, respectively.**


(a) ML1 (Geolife) (b) ML1 (T-Drive) (c) ML1 (OSM)

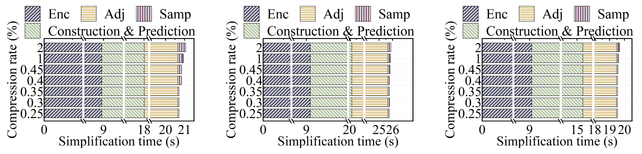

(d) ML2 (Geolife) (e) ML2 (T-Drive) (f) ML2 (OSM)

**Figure 8: Overall simplification time of MLSimp, including the time spent on each component. ML1 and ML2 are as Fig. 7.**

remains minimal, as MLSimp samples directly based on adjusted importance without requiring iterative sampling.

## 5.4 Model Analysis

We evaluate the training cost of MLSimp on Geolife. We compare MLSimp with the learning-based lightweight model S3. Since MLSimp iteratively trains GNN-TS and Diff-TS, we measure the overall

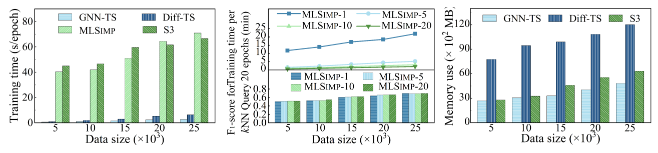

(a) Training time (b) Warm-up epochs (c) Memory budget

**Figure 9: Model analysis of MLSimp.**

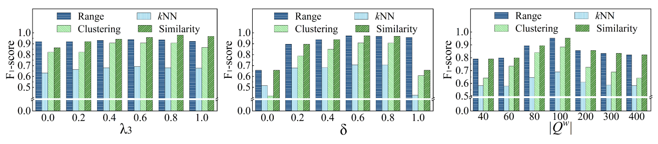
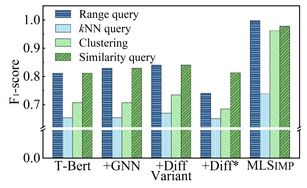

(a) The impact of $\lambda_3$ (b) The impact of $\delta$ (c) The impact of $|Q^w|$

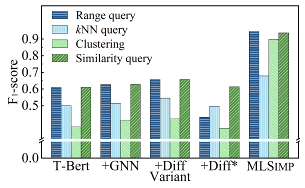
**Figure 10: Impact of hyperparameters.**

training time for MLSimp for one epoch of the mutual learning. The results are shown in Fig. 9 (a). Since the simplification of S3 is based on the Seq2Seq model, which requires iterative encoding of each GPS point, the training time of S3 is substantially higher than those of GNN-TS and Diff-TS. However, the one-epoch training time of MLSimp is higher than that of S3 when the data size is large. To save training costs, we only perform ML after multiple training epochs of GNN-TS and Diff-TS.

Fig. 9 (b) presents (i) the training times for 20 epochs of GNN-TS and Diff-TS (upper part), and (ii) the query $F_1$-scores on the simplified datasets using models trained with different warm-up epochs before ML adjustments (lower part), where MLSimp-i (with $i \in 1, 5, 10, 15$) indicates that GNN-TS and Diff-TS update the ML signal to each other after every $i$ warm-up epochs. At $i = 1$, the models update ML signals after each training epoch, eliminating the need for additional warm-up. As the data size increases, the training time and growth rate of MLSimp-1 are significantly higher than those of models with 5, 10, and 20 warm-up epochs. This is because both GNN-TS and Diff-TS need to perform an inference to generate ML signals. As Diff-TS is a generative model requiring multiple iterations, each inference takes substantial time. Although MLSimp-5, MLSimp-10, and MLSimp-20 reduce the frequency of updating ML information, the query performance of the trained MLSimp-i ($i \in \{1, 5, 10, 15\}$) models is almost identical. The reason

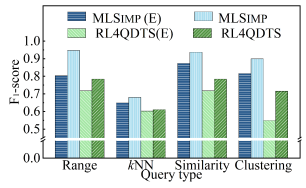


(a) Compression rate = 0.25%

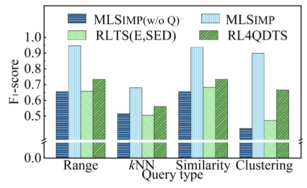


(b) Compression rate = 2%

**Figure 11: Ablation study results.**



(a) Global simplification



(b) Simulated queries

**Figure 12: Case study results.**

may be that, despite the reduced frequency of ML information updates, the training objective remains unchanged, resulting in a similar distribution in the encoding space of the trajectory points in the final GNN-TS. Therefore, in all experiments in Sec. 5, we use MLSimp-20 as the training strategy.

Next, we evaluate the memory use of GNN-TS, Diff-TS, and S3. Since MLSimp uses GNN-TS for simplification, the memory use of GNN-TS is the same as the memory usage of MLSimp during simplification. Fig. 9 (c) shows the results for GNN-TS, Diff-TS, and S3 on Geolife with different data sizes. We observe that the memory use of GNN-TS is slightly lower than that of S3, while the memory use of Diff-TS is significantly higher than that of GNN-TS and S3. This is because Diff-TS contains stacked Transformer layers.

## 5.5 Hyperparameter Study

Hyperparameter $\lambda_3$ is used to adjust the weight of the amplified label in the mutual training loss of GNN-TS. The $F_1$-scores of queries with different $\lambda_3$ are shown in Fig. 10 (a), where $\lambda_3$ ranges from 0 to 1. MLSimp performs best across different queries when $0.6 \leq \lambda_3 \leq 0.8$. Additionally, the impact of $\lambda_3$ on range queries is less significant than on the other three query types. This is because range queries have a lower correlation with the distribution of trajectories.

Hyperparameter $\delta$ is the adjustment weight based on queries. The query performance for different $\delta$ is shown in Fig. 10 (b), where $\delta$ ranges from 0 to 1. When $\delta = 0$, no adjustment is made. When $\delta = 1$, the importance of GPS points is only related to the queries. MLSimp achieves the best performance across different queries when $0.6 \leq \delta \leq 0.8$. This is because the query-based adjustment provides query information to MLSimp, allowing it to consider the importance of both the distribution of trajectories and the queries.

Hyperparameter $|Q^w|$ represents the number of queries used for importance adjustment. The query performance for different values of $|Q^w|$ is shown in Fig. 10 (c), where $|Q^w|$ varies from 40 to 400. When $|Q^w| = 100$, MLSimp achieves the best performance across different queries. When $|Q^w|$ exceeds 100, the $F_1$-scores of all queries decrease as the number of queries increases. Conversely, when $|Q^w|$ is below 100, the $F_1$-scores of all queries decrease as the number of queries decreases. This is because when too many queries are used for adjustment, the region involved in the adjustment increases, and the adjustment cannot focus on popular regions, potentially introducing noise. When too few queries are used, the importance adjustment lacks sufficient attention to popular regions, neglecting points that are truly important for the queries.

## 5.6 Ablation Study

We conduct ablation experiments on Geolife and report the $F_1$-score of five variants in Fig. 11. The five variants are given as follows: (a) "T-Bert" represents a variant that only uses a pre-trained T-Bert to generate trajectory embeddings, followed by sampling based on the importance evaluation in Eq. 6; (b) "+GNN" is a variant that utilizes a GNN for self-supervised training on top of T-Bert to generate trajectory embeddings; (c) "+Diff-TS" is a variant where GNN-TS is trained with Diff-TS for mutual learning; (d) "+Diff*" is the variant where GNN-TS is trained using ML and Diff-TS, but the loss function includes only the ML loss $\mathcal{L}_{ml}$ and excludes the contrastive loss $\mathcal{L}_{con}$ in Eq. 12; (e) "MLSimp" represents the complete model with query-based importance adjustment.

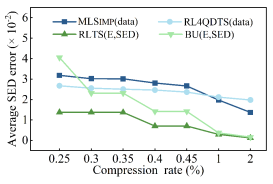
Under both compression rates, GNN-TS, Diff-TS, the mutual learning paradigm, and importance adjustment enhance the performance of queries. Additionally, the performance of the +Diff* variant is consistently lower than the performance of all other variants. The reason is that without the guidance of $\mathcal{L}_{con}$, GNN-TS loses the ability to filter important trajectory points coarsely. The amplified signal generated by Diff-TS, influenced by GNN-TS, cannot guarantee the inclusion of important trajectory points.

## 5.7 Case Study

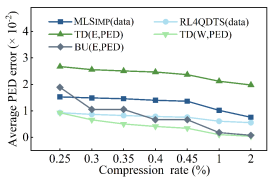
**Impact of global simplification.** Fig. 12 (a) shows the performance of MLSimp and RL4QDTS on Geolife, comparing global simplification to their variants that simplify each trajectory independently (MLSimp(E) and RL4QDTS(E)). All methods are tested across four types of queries at a compression rate of 0.25%. The global versions of MLSimp and RL4QDTS achieve higher $F_1$-scores than their independent counterparts. This is because global simplification adaptively selects the compression rate for each trajectory under a fixed database compression rate.

***Discussion.*** MLSimp(E) outperforms RL4QDTS(E) on similarity-based queries (e.g., $k$NN queries, similarity search, and clustering), even surpassing RL4QDTS. This is because the simplified trajectories of MLSimp(E) retain more global information than those of RL4QDTS(E) at the same trajectory length. MLSimp(E) effectively incorporates global information for each trajectory point during encoding through multiple iterations of information aggregation by GNN-TS, determining the importance of trajectory points based on their distribution. RL4QDTS(E) and RL4QDTS, which only consider local information, retain points based on range queries, focusing on the specific distribution of individual locations and timestamps.

**Impact of simulated queries.** Fig. 12 (b) reports the $F_1$-scores of MLSimp and RL4QDTS and their variants without using simulated queries at a compression rate of 0.25% on four types of queries on Geolife. MLSimp(w/o Q) is a variant of MLSimp that does not use simulated queries for importance adjustment. We select RLTS(E,SED) as a reinforcement learning competitor that does not use simulated queries. The results of MLSimp and RL4QDTS

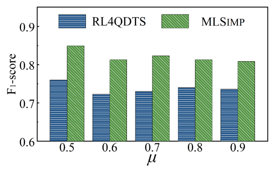


(a) SED error

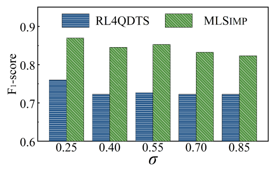


(b) PED error

**Figure 13: Simplification error evaluation on Geolife.**



(a) The impact of $\mu$



(b) The impact of $\sigma$

**Figure 14: Transferability test results.**

with simulated queries consistently outperform MLSimp(w/o Q) and RLTS(E,SED) without simulated queries. This is because MLSimp(w/o Q) lacks adjustments based on query distribution during database simplification. RLTS(E,SED), cannot select trajectory points oriented toward queries.

## 5.8 Simplification Errors Evaluation

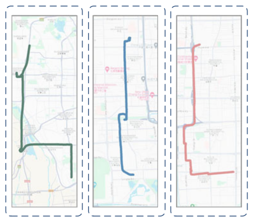
Fig. 13 reports the average SED and PED on Geolife. Since the distribution of queries is irrelevant to the error evaluation, the query-driven methods, MLSimp and RL4QDTS, report the simplification errors using queries generated based on the data distribution. For the error-driven methods, we test the corresponding simplification error results according to the error metrics used by the baseline methods. For both error metrics, the error-driven methods generally outperform the query-driven methods. RLTS(E,SED) achieves the lowest SED for simplified trajectories, while TD(W,PED) achieves the lowest PED for simplified trajectories. The simplification errors of the two query-driven methods are acceptable. MLSimp's simplification error is at most 0.03 higher for SED and at most 0.01 higher for PED compared to the optimal error-driven method. This is because the two query-driven methods generate similar simplified trajectories for similar trajectories, but do not fully support minimizing the errors between the simplified and original trajectories.

## 5.9 Transferability Test

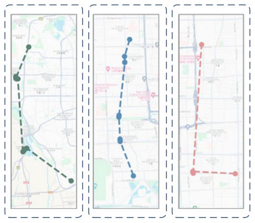
We report the results for range queries with distributions where $\mu$ varies from 0.5 to 0.9 and $\sigma$ from 0.25 to 0.85 in Figs. 14 (a) and (b), respectively. They show that although the performance of both RL4QDTS and MLSimp decline when tested with queries different from those used for simplification, MLSimp consistently outperforms RL4QDTS across different $\mu$ and $\sigma$ values. This is due to MLSimp's use of global information to eliminate semantically redundant points. Moreover, during adjustment, the hyperparameter $\delta$ controls the simplification result without relying solely on the generated queries for simplification, enhancing MLSimp's ability to generalize across different query distributions.

## 5.10 Visualization

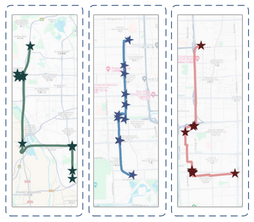
Fig. 15 visualizes the original trajectories and simplified results (with a 2% compression rate) of three trajectories from Geolife.



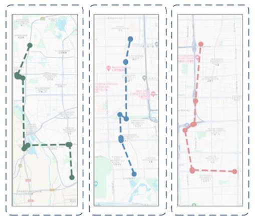
(a) Original trajecotries



(b) GNN-TS without ML



(c) Amplify signal by Diff-TS



(d) GNN-TS with ML

**Figure 15: Simplified trajectory visualization. Circles denote trajectory points, dashed lines connect adjacent points in the simplified trajectory, and stars denote points where the importance is amplified.**

Compared to Fig. 15 (a), the simplified trajectories of GNN-TS without ML (Fig. 15 (b)) have multiple redundant nodes at some turning points and miss some critical turning points. In Fig. 15 (c), Diff-TS identifies several important points in the trajectory during inference that are not retained in Fig. 15 (b). Although the information in amplified signals still contains redundant and non-important points, the GNN-TS with ML incorporates the knowledge extracted by Diff-TS, generating simplified trajectories more similar in shape to the original trajectory, as shown in Fig. 15 (d). This is because the redundancy and non-important points in the GNN-TS without ML have low uniqueness or globality. Even when injected with the amplified signal, their importance remains lower than that of truly important points. Meanwhile, important points that are unsampled in Fig. 15 (b) gain higher importance due to the amplification signal.

## 6 CONCLUSION

We propose a trajectory simplification method called MLSimp, which retains trajectory information tailored for trajectory querying. MLSimp combines two models, GNN-TS and Diff-TS, to simplify trajectories by predicting and adjusting the importance of trajectory points. GNN-TS uses a pre-trained Bert and a GNN to predict the importance of trajectory points, while Diff-TS utilizes a diffusion model to generate simplified trajectories. During training, the two models are used for mutual learning to enhance performance. Experiments on three real-world datasets offer concrete evidence of the effectiveness and efficiency of MLSimp. In future research, it is of interest to extend MLSimp to support online compression of streaming trajectories and road network trajectory compression.

## ACKNOWLEDGMENTS

This research is supported in part by the National Natural Science Foundation of China (U23B2019, 62072083), the Fundamental Research Funds of the Central Universities (N2216017) and the Innovation Fund Denmark project DIREC (9142-00001B).

## REFERENCES

[1] Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150* (2020).

[2] Fernando Berzal and Nicolfás Matín. 2002. Data mining: concepts and techniques by Jiawei Han and Micheline Kamber. In *SIGMOD*. 66–68.

[3] Yanchuan Chang, Jianzhong Qi, Yuxuan Liang, and Egemen Tanin. 2023. Contrastive Trajectory Similarity Learning with Dual-Feature Attention. In *ICDE*. 2933–2945.

[4] Yanchuan Chang, Egemen Tanin, Gao Cong, Christian S Jensen, and Jianzhong Qi. 2024. Trajectory similarity measurement: An efficiency perspective. *PVLDB* 17, 9 (2024), 2293–2306.

[5] Lei Chen, M Tamer Özsu, and Vincent Oria. 2005. Robust and fast similarity search for moving object trajectories. In *SIGMOD*. 491–502.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[7] David H Douglas and Thomas K Peucker. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization* 10, 2 (1973), 112–122.

[8] Ziquan Fang, Yuntao Du, Xinjun Zhu, Danlei Hu, Lu Chen, Yunjun Gao, and Christian S Jensen. 2022. Spatio-temporal trajectory similarity learning in road networks. In *KDD*. 347–356.

[9] Ziquan Fang, Changhao He, Lu Chen, Danlei Hu, Qichen Sun, Linsen Li, and Yunjun Gao. 2023. A lightweight framework for fast trajectory simplification. In *ICDE*. 2386–2399.

[10] Shansan Gong, Mukai Li, Jiangtao Feng, Zhiyong Wu, and Lingpeng Kong. 2023. DiffuSeq: Sequence to Sequence Text Generation with Diffusion Models. In *ICLR*.

[11] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. 855–864.

[12] Mandy Guo, Joshua Ainslie, David Uthus, Santiago Ontanon, Jianmo Ni, Yun-Hsuan Sung, and Yinfei Yang. 2021. LongT5: Efficient text-to-text transformer for long sequences. *arXiv preprint arXiv:2112.07916* (2021).

[13] Yunheng Han, Weiwei Sun, and Baihua Zheng. 2017. COMPRESS: A comprehensive framework of trajectory compression in road networks. *TODS* 42, 2 (2017), 1–49.

[14] John Edward Hershberger and Jack Snoeyink. 1992. Speeding up the Douglas-Peucker line-simplification algorithm. In *SDH*. 134–143.

[15] Cuiying Huo, Di Jin, Yawen Li, Dongxiao He, Yu-Bin Yang, and Lingfei Wu. 2023. T2-GNN: Graph neural networks for graphs with incomplete features and structure via teacher-student distillation. In *AAAI*. 4339–4346.

[16] Jiawei Jiang, Dayan Pan, Houxing Ren, Xiaohan Jiang, Chao Li, and Jingyuan Wang. 2023. Self-supervised trajectory representation learning with temporal regularities and travel semantics. In *ICDE*. 843–855.

[17] Bingqing Ke, Jie Shao, and Dongxiang Zhang. 2017. An efficient online approach for direction-preserving trajectory simplification with interval bounds. In *MDM*. 50–55.

[18] Bingqing Ke, Jie Shao, Yi Zhang, Dongxiang Zhang, and Yang Yang. 2016. An online approach for direction-based trajectory compression with error bound guarantee. In *AP-Web*. 79–91.

[19] Margret Keuper, Bjoern Andres, and Thomas Brox. 2015. Motion trajectory segmentation via minimum cost multicuts. In *ICCB*. 3271–3279.

[20] Thomas N Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*. 1–14.

[21] Lecheng Kong, Jiarui Feng, Hao Liu, Dacheng Tao, Yixin Chen, and Muhan Zhang. 2023. MAG-GNN: Reinforcement Learning Boosted Graph Neural Network. In *NIPS*. 12000–12021.

[22] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. 2007. Trajectory clustering: a partition-and-group framework. In *SIGMOD*. 593–604.

[23] Tianyi Li, Lu Chen, Christian S Jensen, and Torben Bach Pedersen. 2021. TRACE: Real-time compression of streaming trajectories in road networks. *PVLDB* 14, 7 (2021), 1175–1187.

[24] Tianyi Li, Lu Chen, Christian S Jensen, Torben Bach Pedersen, Yunjun Gao, and Jilin Hu. 2022. Evolutionary clustering of moving objects. In *ICDE*. 2399–2411.

[25] Tianyi Li, Ruikai Huang, Lu Chen, Christian S Jensen, and Torben Bach Pedersen. 2020. Compression of uncertain trajectories in road networks. *PVLDB* 13, 7 (2020), 1050–1063.

[26] Jiajun Liu, Kun Zhao, Philipp Sommer, Shuo Shang, Brano Kusy, and Raja Jurdak. 2015. Bounded quadrant system: Error-bounded trajectory compression on the go. In *ICDE*. 987–998.

[27] Jiajun Liu, Kun Zhao, Philipp Sommer, Shuo Shang, Brano Kusy, Jae-Gil Lee, and Raja Jurdak. 2016. A novel framework for online amnesic trajectory compression in resource-constrained environments. *TKDE* 28, 11 (2016), 2827–2841.

[28] Cheng Long, Raymond Chi-Wing Wong, and HV Jagadish. 2013. Direction-preserving trajectory simplification. *PVLDB* 6, 10 (2013), 949–960.

[29] Cheng Long, Raymond Chi-Wing Wong, and HV Jagadish. 2014. Trajectory simplification: On minimizing the direction-based error. *PVLDB* 8, 1 (2014), 49–60.

[30] Pierre-François Marteau and Gildas Ménier. 2009. Speeding up simplification of polygonal curves using nested approximations. *Pattern Analysis and Applications* 12, 4 (2009), 367–375.

[31] Nirvana Meratnia and Rolf A de By. 2004. Spatiotemporal compression techniques for moving point objects. In *EDBT*. 765–782.

[32] Jonathan Muckell, Jeong-Hyon Hwang, Vikram Patil, Catherine T Lawson, Fan Ping, and SS Ravi. 2011. SQUISH: an online approach for GPS trajectory compression. In *COM.Geo*. 1–8.

[33] Jonathan Muckell, Paul W Olsen, Jeong-Hyon Hwang, Catherine T Lawson, and SS Ravi. 2014. Compression of trajectory data: a comprehensive evaluation and new approach. *GeoInformatica* 18 (2014), 435–460.

[34] Aiden Nibali and Zhen He. 2015. Trajic: An effective compression system for trajectory data. *TKDE* 27, 11 (2015), 3138–3151.

[35] Michalis Potamias, Kostas Patroumpas, and Timos Sellis. 2006. Sampling trajectory streams with spatiotemporal criteria. In *SSDBM*. 275–284.

[36] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing* 45, 11 (1997), 2673–2681.

[37] Amílcar Soares Júnior, Bruno Neiva Moreno, Valéria Cesário Times, Stan Matwin, and Lucídio dos Anjos Formiga Cabral. 2015. GRASP-UTS: an algorithm for unsupervised trajectory segmentation. *IJGIS* 29, 1 (2015), 46–68.

[38] Yumeng Song, Yu Gu, Tianyi Li, Jianzhong Qi, Zhenghao Liu, Christian S. Jensen, and Ge Yu. 2024. CHGNN: A Semi-Supervised Contrastive Hypergraph Learning Network. *TKDE* 36, 9 (2024), 4515–4530.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *NIPS* 30 (2017).

[40] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

[41] Jiaqi Wang, Tianyi Li, Anni Wang, Xiaoze Liu, Lu Chen, Jie Chen, Jianye Liu, Junyang Wu, Feifei Li, and Yunjun Gao. 2023. Real-Time Workload Pattern Analysis for Large-Scale Cloud Databases. *PVLDB* 16, 12 (2023), 3689–3701.

[42] Sheng Wang, Zhifeng Bao, J Shane Culpepper, and Gao Cong. 2021. A survey on trajectory data management, analytics, and learning. *CSUR* 54, 2 (2021), 1–36.

[43] Tongzhou Wang and Phillip Isola. 2020. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *ICML*. 9929–9939.

[44] Zheng Wang, Cheng Long, and Gao Cong. 2021. Trajectory simplification with reinforcement learning. In *ICDE*. 684–695.

[45] Zheng Wang, Cheng Long, Gao Cong, and Christian S Jensen. 2023. Collectively Simplifying Trajectories in a Database: A Query Accuracy Driven Approach. *arXiv preprint arXiv:2311.11204* (2023).

[46] Zheng Wang, Cheng Long, Gao Cong, and Qianru Zhang. 2021. Error-bounded online trajectory simplification with multi-agent reinforcement learning. In *KDD*. 1758–1768.

[47] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive Representation Learning on Temporal Graphs. In *ICLR*. 1–19.

[48] Yuanyuan Yao, Dimeng Li, Hailiang Jie, Hailiang Jie, Tianyi Li, Jie Chen, Jiaqi Wang, Feifei Li, and Yunjun Gao. 2023. SimpleTS: An efficient and universal model selection framework for time series forecasting. *PVLDB* 16, 12 (2023), 3741–3753.

[49] Dongxiang Zhang, Mengting Ding, Dingyu Yang, Yi Liu, Ju Fan, and Heng Tao Shen. 2018. Trajectory simplification: an experimental study and quality analysis. *PVLDB* 11, 9 (2018), 934–946.

[50] Ying Zhang, Tao Xiang, Timothy M Hospedales, and Huchuan Lu. 2018. Deep mutual learning. In *CVPR*. 4320–4328.

[51] Silin Zhou, Jing Li, Hao Wang, Shuo Shang, and Peng Han. 2023. GRLSTM: trajectory similarity computation with graph-based residual LSTM. In *AAAI*. 4972–4980.