GoodTP: An Effective Data Selection Framework for Enhancing Trajectory Similarity Learning via Monte Carlo Tree Search

HAITAO YUAN, Nanyang Technological University, Singapore
GAO CONG, Nanyang Technological University, Singapore

Trajectory similarity computation is fundamental to spatial data management. Recent representation learning methods have advanced similarity computation through neural embeddings, yet they rely on naive random sampling of training data, overlooking strategic data selection that determines learning effectiveness. Existing data selection paradigms are not suitable due to their requirement for labeled trajectory pairs with prohibitive cost. We present GoodTP, a novel framework that addresses challenges in data selection for trajectory similarity learning. First, the framework reformulates the expensive pair selection into iterative trajectory selection that dynamically constructs and evaluates pairs, eliminating exhaustive pairwise similarity pre-computation. Second, we design a hierarchical clustering tree that organizes trajectories across multiple granularities. We model selection as tree sampling optimization via Monte Carlo Tree Search (MCTS), efficiently navigating the exponential policy space. Third, we bridge the local-to-global optimization gap through two complementary mechanisms: (1) An effective learned policy based on Upper Confidence Bound (UCB) balances exploration-exploitation with provable convergence guarantees. (2) Cache-enhanced bilevel optimization learns instance-specific weights that align new pairs with accumulated global pairs while jointly optimizing model parameters. Extensive experiments across representative trajectory similarity models on real-world datasets demonstrate that GoodTP achieves substantial performance improvements.

CCS Concepts: • **Information systems** → **Information integration**; **Spatial-temporal systems**.

Additional Key Words and Phrases: Trajectory Learning; Data Selection; Similarity Computation

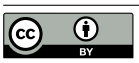
ACM Reference Format:
Haitao Yuan and Gao Cong. 2026. GoodTP: An Effective Data Selection Framework for Enhancing Trajectory Similarity Learning via Monte Carlo Tree Search. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 190 (June 2026), 27 pages. https://doi.org/10.1145/3802067

1 Introduction

Trajectory similarity computation is a fundamental cornerstone of spatial data management [13, 14, 21, 40, 42], enabling critical applications including trajectory clustering [4, 18, 33, 50], similarity-based querying [12, 20, 38, 53], and intelligent traffic optimization [34, 37, 54–56]. While classical point-matching metrics such as DTW [1, 51], EDR [11], and Hausdorff distance [2] provide theoretical foundations, their super-quadratic time complexity creates prohibitive computational bottlenecks for large-scale trajectory datasets.

To address these scalability challenges, recent advances have adopted representation learning approaches [6, 8, 28, 45, 46, 48, 49]. These methods leverage neural network models to map trajectories to vector embeddings and measure similarity via efficient distance metrics (e.g., Euclidean distance) in the embedding space. The models are trained using supervised learning on trajectory

Authors' Contact Information: Haitao Yuan, Nanyang Technological University, Singapore, Singapore, haitao.yuan@ntu.edu.sg; Gao Cong, Nanyang Technological University, Singapore, Singapore, gaocong@ntu.edu.sg.



ACM 2836-6573/2026/6-ART190
https://doi.org/10.1145/3802067

pairs labeled with ground-truth similarities (e.g., DTW). However, **a critical yet overlooked aspect is how training trajectory pairs are selected**. Given N trajectories, one could construct $O(N^2)$ possible pairs. When N is large (e.g., millions), training on all pairs becomes computationally expensive, especially in streaming settings. Moreover, not all pairs contribute equally to learning. Training on low-utility pairs (i.e., identical or extremely dissimilar trajectories) cannot improve the model's ability to capture meaningful similarity patterns. However, existing learning methods [9] rely on naive random sampling without identifying which trajectory pairs are most valuable for learning.

Traditional data selection methods for tasks such as data classification fall into two categories. The first consists of **selection-before-training** approaches, which identify core training subsets using static metrics computed prior to model training. These methods typically rely on gradient-based computations [24, 44] or clustering-based gradient approximations [7] to determine which data samples are most informative for training. While computationally efficient, these methods cannot adapt to evolving model capabilities during training. The second category comprises **Iterative dynamic selection** methods [26, 32, 61], which continuously adapt selection strategies through active/curriculum learning that optimizes training data round-by-round. Despite their success on traditional tasks, three key challenges remain unaddressed for trajectory similarity learning:

Challenge 1 (C1): The Prohibitive Cost of Pair Labeling. Traditional data selection methods rely on labeled training data to guide selection decisions. However, obtaining ground-truth similarity labels for all trajectory pairs is prohibitively expensive. For 1 million trajectories, this means approximately 0.5 trillion pairs. When using accurate similarity metrics like DTW with super-linear complexity, pre-computing all pairs becomes highly expensive. *Consequently, how can we identify high-utility training pairs without exhaustively pre-computing ground-truth similarities?*

Challenge 2 (C2): The Exponential Policy Space. Traditional data selection evaluates samples independently with $O(N)$ complexity. However, trajectory similarity learning depends on *relationships between trajectories*, where each trajectory's quality must be assessed based on its interactions with other trajectories through the pairs it forms. While pairwise interactions scale as $O(N^2)$, the core challenge lies in selecting subsets of trajectories whose joint interactions determine training quality. The number of such subsets grows exponentially with the number of trajectories, making exhaustive search intractable. *Therefore, how can we efficiently search the exponential policy space while accounting for correlations among trajectories?*

Challenge 3 (C3): The Local-to-Global Optimization Gap. Trajectory informativeness evolves during training, making data selection inherently dynamic. However, existing methods optimize only at the local level, meaning trajectory selection within each round, rather than at the global level across the entire training process. Static approaches use fixed metrics that cannot adapt to changing model needs [44], while iterative methods optimize each round independently without considering cumulative effects across the training process [26, 61]. This greedy round-by-round selection ignores how early-round choices constrain later opportunities, resulting in suboptimal overall training distributions. *Therefore, how can we bridge the gap between locally greedy decisions and globally optimal selection?*

Our Solution: We present **GoodTP**, a novel framework that addresses three challenges:

Addressing C1: Rather than directly selecting pairs from the expensive $O(n^2)$ space, we reformulate the problem by converting it into a trajectory selection task. Our framework first selects informative trajectories, then dynamically constructs training pairs from these selected trajectories, and finally evaluates the quality of constructed pairs to provide feedback for trajectory selection decisions. This trajectory-centric paradigm elegantly sidesteps the need to pre-compute ground-truth similarities across all pairs.

Table 1. Summary of Notations

Symbol	Description
$T/T^a/T^b$, p_i, $\mathcal{T}$	Trajectory, trajectory point, trajectory dataset
$\|T\|$, N	Number of points in T, number of trajectories
$f^{a,b}$, g_θ	Similarity score, learning model
$\mathcal{TP}_{\text{train}}/\mathcal{TP}_{\text{val}}/\mathcal{TP}$, B	Training/validation/cached pairs, budget
$\mathbf{f}(T)$, $f_1 \dots f_7$, $\bar{p}$	Feature vector, seven features, trajectory centroid
$\mathcal{I}$, $C[j]$, K	Cluster tree index, cluster, branching factor
$\mathcal{T}_C$, $\mathcal{S}$	trajectory subset in the cluster C, Selected subset
$\boldsymbol{\alpha}$, π	Sampling strategy, traversal policy
$\hat{v}/\bar{v}/v'$, a_v, $\boldsymbol{r}$	Action/expansion/data node, action, allocation ratios
b, n, R, $U(v)$	Cumulative reward, visit count, reward, UCT utility
$w_i/w_{i,j}$, $\ell_{i,j}(\theta)$	Trajectory/pair weight, pair loss
$\mathcal{L}_{\text{train}}/\mathcal{L}_{\text{val}}$, α, β, I_t	Train/val loss, learning rates, iterations

Addressing C2: To tackle the exponential policy space challenge, we design a hierarchical clustering tree that organizes trajectories across multiple levels through top-down recursive partitioning. This structure naturally groups correlated trajectories into manageable clusters and reduces the selection space by enabling cluster-level operations. We then model the trajectory selection problem as an optimization problem of sampling from this tree structure, which allows us to leverage Monte Carlo Tree Search (MCTS) [5] for efficient navigation. Our heterogeneous policy tree contains *action nodes* that decide whether to sample from a cluster or explore its sub-clusters, and *expansion nodes* that determine optimal sampling ratios across sub-clusters. This MCTS-based approach dramatically accelerates sampling efficiency by intelligently pruning the exponential search space through strategic tree traversal.

Addressing C3: To address the gap between per-round selection and overall training objectives, we introduce two complementary mechanisms. First, we adopt a policy learning strategy based on Upper Confidence Bound (UCB) [22] to guide trajectory selection across rounds. Instead of optimizing each round independently, this policy leverages feedback from previous rounds to balance exploration and exploitation, enabling consistent improvement over the entire training process with theoretical convergence guarantees. Second, we introduce a cache-enhanced weighted learning scheme to maintain global consistency. We store high-quality trajectory pairs from previous rounds and use them as a reference distribution. For newly constructed pairs, we solve a bilevel optimization problem where the upper level learns weights to align new pairs with the cached distribution, and the lower level updates model parameters using these weights.

In summary, we make the following contributions:

- We integrate trajectory selection into trajectory similarity learning, establishing an effective trajectory selection optimization framework for the learning. (Sec. 2 and 3)
- We design a hierarchical clustering index and model selection as sampling policy optimization over clustering trees, providing an effective MCTS-based policy learning method with theoretical guarantees. (Sec. 4 and 5)
- We incorporate a cache-enhanced weighting mechanism with bilevel online optimization to jointly reweight pairs and learn similarity model parameters. (Sec. 6)
- We demonstrate significant improvements across four representative trajectory similarity models on three real-world datasets, validating our approach's effectiveness. (Sec. 7)

2 Preliminary

2.1 Basic Concepts

Trajectory. A trajectory $T = [p_1, \dots, p_{|T|}]$ is a sequence of GPS-sampled points. Each point $p_i = (x_i, y_i)$ or $p_i = (x_i, y_i, t_i)$ represents spatial coordinates (latitude, longitude) with an optional

temporal dimension. Following work [8, 48], we adopt 2-dimensional points for simplicity, noting that extension to 3-dimensional trajectories is straightforward.

Trajectory Similarity. Given two trajectories $T^a = [p_1^a, \ldots, p_{|T^a|}^a]$ and $T^b = [p_1^b, \ldots, p_{|T^b|}^b]$, the similarity between T^a and T^b is quantified using distance metrics that compare their respective point sequences $[p_1^a, \ldots, p_{|T^a|}^a]$ and $[p_1^b, \ldots, p_{|T^b|}^b]$. Common metrics include DTW[51], Hausdorff[2], LCSS[39], and Fréchet[16]. For simplicity, we employ DTW as our primary similarity due to its widespread adoption in trajectory analysis applications.

$$
\begin{aligned}
f_{\mathbf{dtw}}^{a,b} &= DTW[p_{|T^a|}^a, p_{|T^b|}^b] \\
DTW[p_i^a, p_j^b] &= \mathbf{dist}(p_i^a, p_j^b) + min\{DTW[p_{i-1}^a, p_j^b], \\
&\qquad DTW[p_i^a, p_{j-1}^b], DTW[p_{i-1}^a, p_{j-1}^b]\}
\end{aligned} \tag{1}
$$

where $\mathbf{dist}(p_i^a, p_j^b)$ is the distance between p_i^a and p_j^b which can be the Euclidean distance or another suitable metric depending on the point definition.

Trajectory Similarity Learning. Computing exact trajectory distances has $O(|T|^2)$ complexity, where $|T|$ denotes the number of trajectory points, making it computationally prohibitive for large-scale applications. Trajectory similarity learning addresses this challenge by training lightweight models to approximate similarity computations. Given a parameterized model $g_\theta(\cdot,\cdot)$, the learning objective is formulated as: $argmin_\theta \sum_{\langle T^a, T^b, f^{a,b}\rangle \in \mathcal{TP}_{\text{train}}} loss(g_\theta(T^a, T^b), f^{a,b})$, where $\mathcal{TP}_{\text{train}}$ represents the training dataset comprising trajectory pairs with ground-truth similarity labels, and $loss(\cdot,\cdot)$ is the loss function that measures the discrepancy between predicted and ground-truth similarities. Table 1 summarizes the main notations.

2.2 Problem Formulation

Data Selection for Trajectory Similarity Learning (TSL). Given a large collection of trajectories $\mathcal{T}$, constructing training data for TSL by enumerating all possible trajectory pairs is computationally prohibitive. With N trajectories, the total number of potential training pairs is N^2, which becomes computationally expensive for large datasets. Therefore, we must strategically sample a subset of trajectory pairs to construct an efficient training dataset. The data selection problem addresses this challenge by identifying a representative subset of trajectory pairs for which ground-truth similarity scores can be computed using the actual similarity function f. This process yields a training set $\mathcal{TP}_{\text{train}} \subseteq \mathcal{T} \times \mathcal{T}$ that is used to train the trajectory similarity model g_θ, where $|\mathcal{TP}_{\text{train}}| \ll N^2$.

Data Selection Optimization (DSO) in Learning Pipeline. For simplicity, we model data selection as a single process within the learning pipeline, which can be formalized as $\mathcal{TP}_{\text{val}} = F_\gamma(\mathcal{T})$, where F_γ represents the data selection framework parameterized by γ. The optimization objective is to determine the optimal configuration of F_γ such that the resulting model g_θ achieves maximum performance on the test data. We impose a constraint B on the number of selected training samples. This optimization problem can be formalized as follows:

Definition 1 (DSO for TSL). *Given a trajectory set $\mathcal{T}$, a ground-truth similarity function f, a training budget B (maximum number of trajectory pairs), a test dataset $\mathcal{TP}_{test}$, a trajectory similarity learning model g_θ. The data selection problem seeks to design a selection framework F_γ that retrieves an optimal subset of trajectory pairs from $\mathcal{T} \times \mathcal{T}$ to minimize the validation loss. Formally:*

$$
\begin{aligned}
\arg\min_{\gamma} \quad & \sum_{\langle T_{test}^a, T_{test}^b, f_{val}^{a,b}\rangle \in \mathcal{TP}_{test}} loss(g_{\theta(\gamma)}(T_{test}^a, T_{test}^b), f_{val}^{a,b}) \\
s.t. \quad & \theta(\gamma) = \arg\min_{\theta} \sum_{\langle T^a, T^b, f^{a,b}\rangle \in F_\gamma(\mathcal{T})} loss(g_\theta(T^a, T^b), f^{a,b}) \\
& |F_\gamma(\mathcal{T})| = B
\end{aligned}
$$

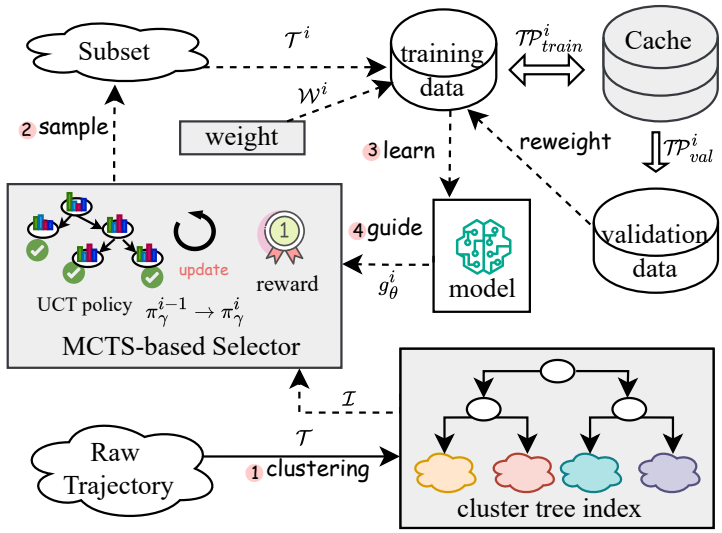


Fig. 1. The Framework of Our Proposed **GoodTP**

where $\theta(\gamma)$ represents the model parameters obtained by training on the selected data based on the selection framework F_γ.

3 Framework Overview

Design Rationale. Our method aligns with the iterative dynamic paradigm due to its superior adaptability to the evolving demands of trajectory similarity learning. Our framework continuously re-evaluates and selects training data based on the trajectory embedding model's current state. This iterative refinement ensures that the most informative trajectory pairs are consistently utilized, steering the model away from suboptimal solutions and toward more robust similarity representations.

Framework Architecture. As illustrated in Figure 1, our framework **GoodTP**, iteratively refines the training set for trajectory similarity learning. The core components are detailed below.

1) Hierarchical Trajectory Clustering. A naive approach to optimizing the training set would involve evaluating each trajectory individually. However, this is computationally infeasible for large-scale datasets. To overcome this scalability challenge, our framework operates on clusters of trajectories rather than individual instances. Specifically, we employ hierarchical clustering to partition the entire trajectory pool $\mathcal{T}$ based on pre-defined features. This method constructs a tree-based index, denoted as $\mathcal{I}$, which naturally supports an efficient, coarse-to-fine selection strategy. Within this index, each leaf node corresponds to a fine-grained cluster of similar trajectories. A non-leaf (internal) node represents a super-cluster formed by the union of the clusters of its children. Consequently, the root of the tree encapsulates the entire trajectory pool $\mathcal{T}$. This hierarchical structure allows our selection mechanism to efficiently navigate the search space by pruning or selecting entire groups of trajectories at once.

2) MCTS-based Sampling. Given the constructed cluster tree index, we iteratively select optimal trajectories using a selector to generate training data for each round. However, exhaustively evaluating all possible subsets of trajectories is computationally infeasible, as the number of candidate subsets grows exponentially with the number of clusters. To address this challenge, we develop an MCTS-based selector leveraging Monte Carlo Tree Search [5] to efficiently navigate the combinatorial selection space. We reformulate trajectory selection as a sampling optimization task

over clusters, determining which branches to explore and their sampling ratios—naturally forming a tree search policy optimization problem where nodes represent clusters and edges represent sampling decisions. We adopt the UCT (Upper Confidence bounds applied to Trees [22]) algorithm to derive optimal sampling strategies, balancing exploration of promising but under-sampled clusters with exploitation of previously high-quality clusters. Finally, we apply the generated sampling strategy to extract the selected trajectory subset $\mathcal{T}^i$ for the i-th round of training.

3) Cache-enhanced Weighted Model Learning. Given the dynamically selected trajectories, we construct training data comprising trajectory pairs and their corresponding similarity scores. To address the quadratic complexity of similarity computation, we incorporate a caching mechanism for efficiency. Specifically, we first check whether similarities for current trajectory pairs have been computed in previous rounds; if not, we compute them directly and cache the results for future use, yielding training data $\mathcal{TP}^i_{train}$ for round i. However, since the policy performs local optimization each round, the selected trajectories may not capture the global distribution of the entire trajectory pool. To address this bias, we sample validation data $\mathcal{TP}^i_{val}$ to reweight the training samples. We initialize weights $\mathcal{W}^i = \mathbf{1} \in \mathbb{R}^{|\mathcal{T}^i|}$ and apply a reweighting method to adjust $\mathcal{W}^i$ during similarity model learning.

4) Policy Updating with Model Guidance. At the end of each round, we obtain the updated model g^i_θ and leverage it to provide feedback for policy updates. Specifically, we use the model to calculate rewards when applying the UCT algorithm, updating the policy parameters from $\pi_{\gamma^{i-1}}(\mathcal{I})$ to $\pi_{\gamma^i}(\mathcal{I})$.

4 Hierarchical Clustering

Motivation. Given the complexity of trajectory data, clustering typically involves two steps: defining trajectory similarity or distance metrics (e.g., DTW [51], Hausdorff [2] and EDR [11]), and applying traditional clustering algorithms such as K-means [30] or DBSCAN [17] to obtain clustering results. However, existing approaches have significant limitations for our task. Current similarity definition methods fall into two problematic categories: spatial distance-based approaches [27, 29, 41] that are task-specific, lack generalizability, and often have quadratic computational complexity resulting in low efficiency; and representation learning-based methods [43, 50] that are typically unsupervised and lack interpretability, making clustering rationale unclear. Additionally, most existing methods require predefining a fixed number of clusters, which directly determines the trajectory partition used for selection. This design enforces a fixed clustering granularity, limiting category adaptability and restricting the flexibility of the selection space for downstream applications.

To address these limitations, we design a novel clustering approach with two key improvements: first, we develop interpretable trajectory features tailored to our task, enabling similarity-based clustering that is both meaningful and explainable (Sec. 4.1); second, we implement a flexible hierarchical clustering framework that allows downstream applications to adaptively combine clusters at different granularity levels according to their specific needs, rather than being constrained by fixed category numbers (Sec. 4.2).

4.1 Trajectory Feature Extraction

Prior work [27] demonstrates that trajectory similarity encompasses orthogonal dimensions of spatial alignment and temporal motion patterns. Therefore, we design seven features to capture complementary trajectory properties across spatial distribution and movement dynamics, following established practices in trajectory mining literature [13, 15]. For instance, the spatial span (f_1), center distance features (f_2, f_3) and the normalized length (f_4) draw inspiration from spatial distance

metrics [39] and spatial shape representation analysis [57], while the directional changes (f_5, f_6) and step length (f_7) adopt movement pattern analysis from [60].

To formally define the proposed features, we introduce the following notations. Given a trajectory $T = \{p_i\}_{i=1}^{|T|}$, where $p_i \in \mathbb{R}^d$ denotes a point in a d-dimensional space, we define the coordinate-wise minimum and maximum points of T as $p_{\min} = (\min_i\{p_i^{(1)}\}, \ldots, \min_i\{p_i^{(d)}\})$ and $p_{\max} = (\max_i\{p_i^{(1)}\}, \ldots, \max_i\{p_i^{(d)}\})$, which correspond to the corners of the trajectory's axis-aligned bounding box. We further define a dataset-level normalization constant, $S_{\max} = \max_{T' \in \mathcal{T}} \|\mathbf{p}'_{\max} - \mathbf{p}'_{\min}\|_2$, which corresponds to the maximum bounding-box diagonal length over the trajectory dataset $\mathcal{T}$. In addition, we introduce the trajectory centroid as $\bar{p} = \frac{1}{|T|}\sum_{i=1}^{|T|} p_i$. Notably, each feature is normalized to $[0, 1]$ to ensure balanced contribution during clustering.

(1) Spatial Distribution Features. These features characterize how trajectory points are distributed in geographic space, capturing both extent and concentration patterns. They are defined as follows:

Spatial Span: $f_1 = \|p_{\max} - p_{\min}\|_2 / S_{\max}$ measures the ℓ_2 norm of the trajectory's bounding box diagonal, quantifying overall geographic coverage.

Mean Center Distance and Variance: Let $d_i = \|p_i - \bar{p}\|_2$ denote the distance from point p_i to the trajectory centroid, and $D_{\max} = \max_j d_j$ be the maximum center distance within T. We define $f_2 = \mathbb{E}[d_i]/D_{\max}$ and $f_3 = \mathrm{Var}[d_i]/D_{\max}^2$, where $\mathbb{E}[\cdot]$ and $\mathrm{Var}[\cdot]$ denote the sample mean and variance over all trajectory points, respectively. These two features jointly characterize spatial concentration and dispersion patterns.

(2) Movement Pattern Features. These features capture dynamic behavioral aspects, focusing on directional consistency and movement regularity. In particular, they are defined as follows:

Normalized Trajectory Length: $f_4 = |T|/L_{\max}$ measures trajectory density, where $|T|$ is the number of trajectory points and $L_{\max}$ is the dataset maximum length.

Mean Direction Change and Variance: Let $\theta_i = \arccos\left(\frac{\mathbf{v}_i \cdot \mathbf{v}_{i+1}}{\|\mathbf{v}_i\|_2 \|\mathbf{v}_{i+1}\|_2}\right)$ denote the turning angle between consecutive movement vectors $\mathbf{v}_i = p_{i+1} - p_i$. We define $f_5 = \mathbb{E}[\theta_i]/\pi$ and $f_6 = \mathrm{Var}[\theta_i]/\pi^2$, which jointly characterize turning behavior and movement predictability.

Average Step Length: This feature is denoted as $f_7 = \mathbb{E}[\|p_{i+1} - p_i\|_2]/S_{\max}$, which characterizes typical spatial displacement, distinguishing fast-moving from slow-moving trajectories.

The 7-dimensional feature vector $\mathbf{f}(T) = [f_1, f_2, \ldots, f_7]$ provides a compact yet comprehensive representation suitable for scalable clustering algorithms while preserving interpretability for domain experts. Importantly, our feature design is agnostic to the **GoodTP** framework itself. The hierarchical clustering module operates on any feature vector representation, making the framework flexible and extensible to accommodate alternative or richer feature sets tailored to specific application domains.

4.2 Cluster Tree Index Construction

Based on the extracted features for each trajectory, denoted as $\{\mathbf{f}(T_1), \mathbf{f}(T_2), \cdots\}$, we iteratively incorporate the K-means technique to cluster trajectories in a top-down hierarchical manner, leading to the construction of cluter tree index $\mathcal{I}$. Specifically, our framework operates through the following recursive process:

(1) Initial Clustering: We first apply K-means clustering to the complete feature set $\{\mathbf{f}(T_i)\}_{i=1}^{N}$, partitioning all trajectories into K initial clusters $\{C[1], C[2], \ldots, C[K]\}$. Each cluster $C[j]$ contains trajectories with similar movement characteristics as captured by our 7-dimensional feature representation.

(2) Recursive Subdivision: For each cluster $C[j]$ at level ℓ, we recursively apply K-means clustering to its constituent trajectory features, generating K_j subclusters $\{C[j, 1], C[j, 2], \ldots, C[j, K_j]\}$ at

level $\ell + 1$. This process continues until a termination criterion is met (e.g., minimum cluster size, maximum depth, or within-cluster homogeneity threshold).

Adaptive Granularity Selection: Unlike traditional approaches that produce a fixed number of clusters, our framework enables adaptive cluster selection through three mechanisms:

(1)*Level-based Selection:* Users can extract clustering results at any hierarchical level ℓ, yielding different numbers of clusters depending on the desired granularity.

(2) *Mixed-level Composition:* Different subtrees can be selected at varying depths, allowing fine-grained control over specific trajectory groups while maintaining coarser granularity for others.

For example, applications can select clusters $\{C[1], C[2], C[3]\}$ or $\{C[1,1], \cdots, C[1,K], C[2], C[3]\}$. Consequently, downstream applications can dynamically select the most appropriate clustering granularity without requiring algorithmic re-execution, significantly enhancing both computational efficiency and analytical flexibility. This adaptive mechanism fundamentally differs from traditional approaches that lock the selection policy into a predetermined partition.

5 MCTS-based Trajectory Selection

In this section, we present our approach to trajectory selection through hierarchical cluster sampling. We first model the individual trajectory selection problem as a node traversal policy optimization problem over a hierarchical tree index (Sec. 5.1). Subsequently, we describe our MCTS-based policy learning method for solving this problem with a theoretical guarantee (Sec. 5.2).

5.1 Cluster Node Traversal Policy Modeling

The trajectory selection problem can be naturally formulated as a combinatorial optimization task. Given a trajectory dataset $\mathcal{T} = \{T_1, T_2, \ldots, T_N\}$, an objective function $O : 2^{\mathcal{T}} \to \mathbb{R}$, and a budget constraint requiring exactly B trajectories where $B \leq N$, the optimization problem is:

$$\mathcal{S}^* = \arg\max_{\mathcal{S} \subseteq \mathcal{T}} O(\mathcal{S}) \quad \text{subject to} \quad |\mathcal{S}| = B \tag{2}$$

where $O(\mathcal{S})$ is evaluated by a downstream learning model. However, this formulation suffers from exponential complexity, as it requires exploring $\binom{N}{B}$ possible combinations.

Hierarchical Cluster Sampling Formulation. To address this computational challenge, we leverage the hierarchical cluster tree structure to reformulate the problem. Let $\mathcal{I}$ denote our hierarchical cluster tree. Within this tree, we can extract different clustering levels, each representing a different granularity of trajectory grouping. For a given clustering level, let $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ denote the set of cluster nodes, where each cluster C_i corresponds to a disjoint trajectory subset $\mathcal{T}_{C_i} \subseteq \mathcal{T}$. The clusters form a partition of the trajectory dataset, satisfying:

$$\mathcal{T}_{C_i} \cap \mathcal{T}_{C_j} = \emptyset \quad \text{for } i \neq j, \quad \text{and} \quad \bigcup_{i=1}^{m} \mathcal{T}_{C_i} = \mathcal{T} \tag{3}$$

Based on this clustering structure, we define the sampling strategy as follows:

Definition 2 (Sampling Strategy). *A* sampling strategy *is defined as* $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ *where* $\alpha_i \in [0, 1]$ *represents the sampling ratio for cluster* C_i.

The equivalent cluster-based optimization problem becomes:

$$\boldsymbol{\alpha}^* = \arg\max_{\boldsymbol{\alpha}} O\left(\bigcup_{i=1}^{m} \text{Sample}(\mathcal{T}_{C_i}, \alpha_i)\right) \quad \text{s.t.} \quad \sum_{i=1}^{m} \alpha_i |\mathcal{T}_{C_i}| = B \tag{4}$$

where $\text{Sample}(\mathcal{T}_{C_i}, \alpha_i)$ denotes uniform random sampling of $\lfloor \alpha_i |\mathcal{T}_{C_i}| \rfloor$ trajectories from cluster C_i.

Theoretical Foundation. We now establish the theoretical equivalence between the original trajectory selection problem and our cluster-based reformulation.

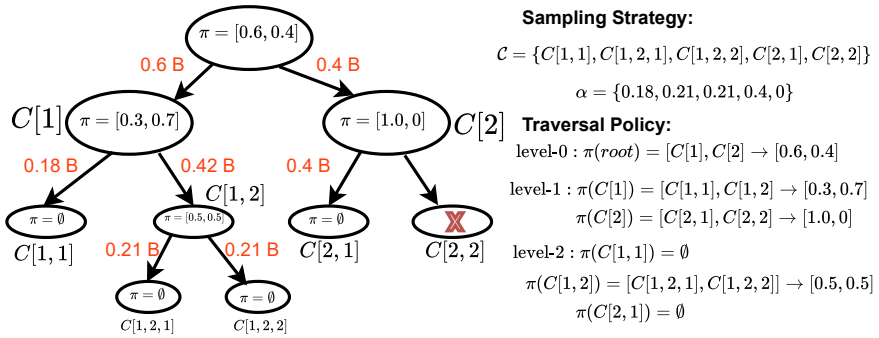


Fig. 2. An Example of Node Traversal Policy ($K = 2$)

Theorem 1 (Sampling Equivalence). *For any trajectory selection objective function O and clustering C satisfying the partition property in Eq.* (3)*, problems defined by Eq.* (2) *and Eq.* (4) *are equivalent in the following sense:*

(1) Every feasible solution $\mathcal{S}$ can be represented by a strategy $\boldsymbol{\alpha}$;
(2) Every sampling strategy $\boldsymbol{\alpha}$ corresponds to a feasible solution $\mathcal{S}$;
(3) Objective function values are preserved under this correspondence.

Proof. We establish equivalence through bijective mapping between solution spaces.
(1) Forward Direction ($\mathcal{S} \rightarrow \boldsymbol{\alpha}$): Given any feasible solution $\mathcal{S} \subseteq \mathcal{T}$ with $|\mathcal{S}| = B$, we construct the corresponding sampling strategy by defining $\alpha_i = \frac{|\mathcal{S} \cap \mathcal{T}_{C_i}|}{|\mathcal{T}_{C_i}|}$. This construction ensures: (i) $\alpha_i \in [0, 1]$ by definition since $\mathcal{S} \cap \mathcal{T}_{C_i} \subseteq \mathcal{T}_{C_i}$, (ii) the constraint $\sum_{i=1}^{m} \alpha_i |\mathcal{T}_{C_i}| = \sum_{i=1}^{m} |\mathcal{S} \cap \mathcal{T}_{C_i}| = |\mathcal{S}| = B$ is satisfied due to the partition property, and (iii) objective preservation follows since $\mathcal{S} = \bigcup_{i=1}^{m} (\mathcal{S} \cap \mathcal{T}_{C_i})$.
(2) Backward Direction ($\boldsymbol{\alpha} \rightarrow \mathcal{S}$): Given any sampling strategy $\boldsymbol{\alpha}$ satisfying the constraint $\sum_{i=1}^{m} \alpha_i |\mathcal{T}_{C_i}| = B$, the solution is $\mathcal{S} = \bigcup_{i=1}^{m} \text{Sample}(\mathcal{T}_{C_i}, \alpha_i)$. The constraint ensures $|\mathcal{S}| = \sum_{i=1}^{m} \lfloor \alpha_i |\mathcal{T}_{C_i}| \rfloor \approx B$ (exact equality when all $\alpha_i |\mathcal{T}_{C_i}|$ are integers), and objective function preservation follows directly from the construction.
(3) Bijectivity: The above two mappings are inverses of each other up to sampling randomness, establishing bijective correspondence between solution spaces. □

Node Traversal Policy. To solve the sampling optimization problem, we introduce a policy π that systematically generates the sampling configuration consisting of selected cluster nodes C and their corresponding sampling ratios $\boldsymbol{\alpha}$. As illustrated in Figure 2, the policy operates on the cluster tree using a breadth-first traversal strategy. The policy framework functions as follows: Starting from the root node, the policy π makes decisions at each node regarding whether to: (1) terminate the search and directly sample from the current node, or (2) continue searching deeper into its children with specified allocation ratios.
(1) Concrete Example: In Figure 2, the execution proceeds as follows:

- *Root Node Policy:* The policy decides to search both children $C[1]$ and $C[2]$ with allocation ratios $[0.6, 0.4]$, distributing the total sampling budget B proportionally.
- *Node $C[1]$ Policy:* The policy further subdivides $C[1]$ into its children $C[1, 1]$ and $C[1, 2]$ with ratios $[0.3, 0.7]$. For $C[1, 1]$, the policy terminates the search, resulting in a final sampling allocation of $\alpha_{C[1,1]} = 0.6 \times 0.3 = 0.18$.
- *Node $C[1, 2]$ Policy:* The policy continues searching $C[1, 2]$, generating two leaf-level nodes $C[1, 2, 1]$ and $C[1, 2, 2]$ with equal allocation ratios $[0.5, 0.5]$. This yields sampling allocations of $\alpha_{C[1,2,1]} = \alpha_{C[1,2,2]} = 0.6 \times 0.7 \times 0.5 = 0.21$.

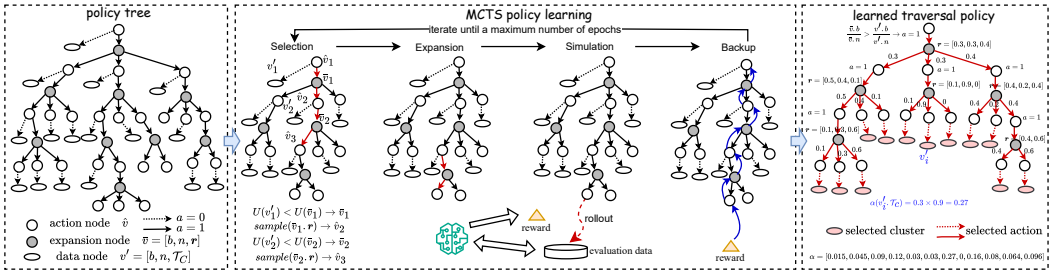


Fig. 3. The Framework Overview of MCTS-based Traversal Policy Learning for Trajectory Sampling

- *Node $C[2]$ Policy:* The policy processes $C[2]$ with allocation ratios $[1.0, 0.0]$ for its children $C[2,1]$ and $C[2,2]$. Since $C[2,2]$ receives zero allocation, it is effectively pruned from the sampling process. Node $C[2,1]$ receives the full allocation $\alpha_{C[2,1]} = 0.4 \times 1.0 = 0.4$.

(2) Policy Formalization: Mathematically, each node v in the cluster tree is associated with a policy function $\pi(v) = (a_v, \boldsymbol{r}_v)$ where:

- $a_v \in \{0, 1\}$ is the binary action: 0 for terminating search and sampling directly from node v, 1 for continuing to search the children of v.
- $\boldsymbol{r}_v = (r_{v,1}, r_{v,2}, \ldots, r_{v,k_v}) \in [0,1]^{k_v}$ specifies the allocation ratios for the k_v children of node v, subject to $\sum_{i=1}^{k_v} r_{v,i} = 1$.

A complete traversal policy π is represented as a sequence of node-level decisions following breadth-first traversal order. For the example in Figure 2, the policy sequence is:

$$\pi = [\underbrace{(1,[0.6,0.4])}_{\text{root}}, \underbrace{(1,[0.3,0.7])}_{C[1]}, \underbrace{(1,[1.0,0])}_{C[2]}, \underbrace{(0,\emptyset)}_{C[1,1]}, \underbrace{(1,[0.5,0.5])}_{C[1,2]}, \underbrace{(0,\emptyset)}_{C[2,1]}, \underbrace{(0,\emptyset)}_{C[1,2,1]}, \underbrace{(0,\emptyset)}_{C[1,2,2]}]$$

where nodes with terminating actions ($a_v = 0$) have empty ratio vectors since no further subdivision is performed.

(3) Optimization Objective: The goal is to learn the optimal policy π^* that maximizes the expected performance of the resulting trajectory sample on a given evaluation function O:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\mathcal{S} \sim \pi(\mathcal{I})}[O(\mathcal{S})] \tag{5}$$

where $\mathcal{S} \sim \pi(\mathcal{I})$ represents the trajectory sample generated by applying policy π to the hierarchical cluster tree $\mathcal{I}$.

5.2 MCTS-based Policy Learning

Based on the policy optimization problem defined in Eq. 5, a brute-force approach would enumerate all possible traversal policies over the cluster tree, which is computationally prohibitive. Therefore, we construct a policy tree to efficiently represent the policy search space and apply Monte Carlo Tree Search (MCTS) for policy learning. The optimized policy is then applied to the cluster tree to generate the final sampling configuration.

Policy Tree Structure. As shown in Figure 3, the policy tree mirrors the hierarchical clustering structure and consists of three types of nodes that work together to represent policy decisions.

(1) Action nodes ($\hat{v}$) correspond to clusters in the hierarchical tree and represent binary decision points. Each action node determines whether to continue subdividing the cluster (action = 1) or terminate and sample directly from it (action = 0), serving as the primary decision points in the policy tree.

(2) Expansion nodes ($\bar{v}$) manage the allocation strategy when an action node chooses to continue subdivision. Each expansion node $\bar{v} = (b, n, \boldsymbol{r})$ contains the cumulative reward b from visiting this node, the visit count n, and the allocation ratio vector $\boldsymbol{r}$ that specifies how to distribute sampling budget among child clusters.
(3) Data nodes (v') represent terminal sampling operations. Each data node $v' = (b, n, \mathcal{T}_C)$ stores the cumulative reward b, visit count n, and the trajectory set $\mathcal{T}_C$ of its corresponding cluster. Data nodes perform uniform random sampling from $\mathcal{T}_C$ and evaluate the sampled trajectories to generate rewards.
Node Relationships. The policy tree structure follows specific connectivity rules. Each action node has two possible children: an expansion node when action $a = 1$ or a data node when action $a = 0$. Each expansion node has exactly k action node children, corresponding to the k sub-clusters in the hierarchical tree. Data nodes are leaf nodes with no children, representing the termination of policy execution. This hierarchical structure enables systematic exploration of the policy space while maintaining computational tractability through the MCTS framework.
MCTS-based Learning. Our learning objective is to generate an optimal traversal policy across the policy tree, where we determine the correct action for each action node and generate appropriate allocation ratios for each expansion node. MCTS [5] is a well-established tree search algorithm that balances exploitation (selecting high-reward paths) and exploration (investigating less-visited paths) when navigating the policy tree. Unlike traditional MCTS applications that seek a single optimal node, our approach leverages the search process to accumulate statistical information at each node, enabling informed policy decisions throughout the entire tree. This accumulated knowledge allows us to generate a comprehensive traversal policy that specifies actions and allocation strategies for all nodes. As shown in Figure 3, our MCTS-based policy learning follows a four-phase iterative process that continues for a predetermined number of epochs.
(1) Selection. The selection phase traverses from the policy tree root to identify a node for expansion. Starting at the root, the process navigates downward by making decisions at each action node $\hat{v}$ based on the UCT (Upper Confidence Bound for Trees [22]) criterion. Each action node $\hat{v}$ has two potential children: an expansion node $\bar{v}$ (representing action = 1) and a data node v' (representing action = 0). The selection strategy at each action node follows a three-case decision process: (i) If both children are unvisited ($v'.n = \bar{v}.n = 0$), one child is selected randomly for exploration and the selection process terminates. (ii) If only one child has been visited, the unvisited child is automatically selected and the selection process terminates. (iii) If both children have been explored, the UCT utility function determines the choice: $U(v) = \frac{v.b}{v.n} + c\sqrt{\frac{\ln(v'.n+\bar{v}.n)}{v.n}}$, where $v.b$ is the cumulative reward, $v.n$ is the visit count, and c is the exploration parameter. The child with higher utility value is selected for further selection. Specifically, when an expansion node $\bar{v}$ is selected, the process continues by choosing among its action node children according to the current allocation ratios $\bar{v}.\boldsymbol{r}$, which are initially set to uniform distribution $\frac{1}{k}$ for k children. When a data node v' is selected, the selection phase terminates and moves directly to simulation. Therefore, the selection continues until reaching either an unvisited node or a data node.

Example 2. As shown in Fig. 3, the process starts at root action node $\hat{v}_1$, where $U(v'_1) < U(\bar{v}_1)$ leads to selecting the expansion node $\bar{v}_1$. Following allocation ratios $\bar{v}_1.\boldsymbol{r}$, the action node $\hat{v}_2$ is chosen. At $\hat{v}_2$, since $U(v'_2) < U(\bar{v}_2)$, expansion node $\bar{v}_2$ is selected, which leads to $\hat{v}_3$ based on $\bar{v}_2.\boldsymbol{r}$. Since $\hat{v}_3$'s children are unvisited, selection terminates.

(2) Expansion. The expansion phase only occurs when selection terminates at an unvisited action node. If selection terminates at a data node, the expansion phase is skipped and the process moves directly to simulation. When expansion is triggered at an action node $\hat{v}$, it creates the appropriate

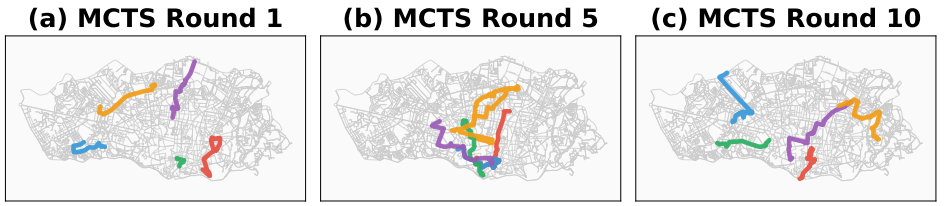


Fig. 4. Evolution of Trajectory Selection Across MCTS Rounds on Porto Map. Each panel shows 5 trajectories.

child node based on a randomly selected action. If the selected action is 0, expansion creates a data node v' with initial values $v'.b = v'.n = 0$ and a trajectory set $\mathcal{T}_C$ corresponding to the cluster, and this data node is selected for simulation. If the selected action is 1, expansion creates an expansion node $\bar{v}$ with initial values $\bar{v}.b = \bar{v}.n = 0$ and uniform allocation ratios $\boldsymbol{r} = [\frac{1}{k}, \frac{1}{k}, \ldots, \frac{1}{k}]$. The expansion node simultaneously creates all k child action nodes corresponding to the sub-clusters, each initialized with $b = n = 0$. After creating these child action nodes, one of them is randomly selected according to the allocation ratios $\boldsymbol{r}$ of the parent expansion node, and this selected action node proceeds to the simulation phase.

(3) Simulation. The simulation phase evaluates the quality of the expanded node by collecting trajectory samples and computing rewards. The simulation process depends on the type of node selected from the expansion phase. (i) If the expanded node is a data node v', simulation directly performs uniform random sampling on the corresponding trajectory set $\mathcal{T}_C$ to collect trajectory samples. (ii) If the expanded node is an action node $\hat{v}$, simulation applies a rollout strategy to explore the subtree rooted at this action node. The rollout process makes random decisions at each action node (randomly choosing between 0 and 1) and assigns random allocation ratios at each expansion node until reaching data nodes. During rollout, multiple data nodes $\{v'_1, v'_2, \ldots, v'_m\}$ may be encountered, and their corresponding trajectory sets $\{\mathcal{T}_{C_1}, \mathcal{T}_{C_2}, \ldots, \mathcal{T}_{C_m}\}$ are collected. Uniform random sampling is then performed on each trajectory set to gather the final trajectory samples.

The collected trajectories are further used to construct an evaluation dataset $\mathcal{TP}_{val}$, which is evaluated using the current trajectory learning model g_θ. The reward is computed as the average loss: $R = \frac{1}{|\mathcal{TP}_{val}|} \sum_i \text{loss}(g_\theta(T_i^a, T_i^b), f_i^{a,b})$, where T_i^a and T_i^b are trajectory pairs and $f_i^{a,b}$ is the ground truth similarity. This formulation is designed to identify challenging training samples: higher prediction errors (larger loss values) indicate more difficult and valuable data for model training, resulting in higher rewards. This guides the search toward sampling strategies that select informative trajectories where the model currently performs poorly, thereby improving training effectiveness.

(4) Backup. The backup phase propagates the computed reward back through the entire traversal path from the simulation node to the root, updating the statistics of all visited nodes. For each node v in the path, the backup process performs different updates depending on the node type. For all nodes, the basic statistics are updated: (1) accumulates the reward by updating $v.b \leftarrow v.b + R$, and (2) increments the visit count by setting $v.n \leftarrow v.n + 1$. For expansion nodes $\bar{v}$, an additional update is performed on the allocation ratios $\bar{v}.\boldsymbol{r}$. The allocation ratios are recomputed based on the average benefits of the child action nodes. Specifically, for each child action node $\hat{v}_i$, the allocation ratio is updated as $\bar{v}.\boldsymbol{r}[i] = \frac{\hat{v}_i.b/\hat{v}_i.n}{\sum_{j=1}^{k} \hat{v}_j.b/\hat{v}_j.n}$, which ensures that child nodes with higher average rewards receive larger allocation ratios in future selections. The accumulated rewards, visit counts, and updated allocation ratios enable accurate estimation of node values for future UCT computations, gradually improving the policy tree's ability to identify promising sampling

strategies. After completing the backup phase, one MCTS iteration is finished, and the process repeats for the next iteration until the predetermined number of epochs is reached.

Applying the learned policy. After the policy has been learned, we traverse the policy tree to generate the final sampling results. Decisions are required only at the action nodes and the expansion nodes. As described earlier, we employ a breadth-first search (BFS) traversal scheme. At an action node $\hat{v}$, we compare the empirical mean rewards of its two child nodes, $\frac{v'.b}{v'.n}$ and $\frac{\bar{v}.b}{\bar{v}.n}$. If the data node provides a lower empirical mean reward (i.e., $\frac{\bar{v}.b}{\bar{v}.n} > \frac{v'.b}{v'.n}$), we choose $a = 1$; otherwise, we set $a = 0$. When $a = 1$, the traversal continues into the corresponding expansion node, where the learned allocation proportion is applied. Finally, we obtain the selected data nodes, each corresponding to a cluster. The sampling proportion of a data node v is determined by the product of the allocation ratios along its ancestral expansion nodes, i.e., $\alpha(v) = \prod_{u \in \text{Ancestors}(v)} \rho(u)$, where $\rho(u)$ denotes the allocation ratio at expansion node u. For instance, as shown in Fig. 3, the sampling proportion of data node v_i' is $\alpha(v_i'.\mathcal{T}.C) = 0.3 \times 0.9 = 0.27$, since its parent has ratio 0.9 and its grandparent has ratio 0.3.

Qualitative Example. Figure 4 visualizes trajectory selection across MCTS rounds on the Porto dataset. *(a) MCTS Round 1*: The policy prioritizes *exploration*, with UCB assigning high bonuses to under-visited clusters to ensure broad geographic coverage. *(b) MCTS Round 5*: The policy shifts toward *exploitation*, discovering *informative hard samples*. Notably, some trajectories share overlapping road segments in the city center while diverging elsewhere. Such partially-overlapping pairs force the model to learn subtle spatial differences, providing high learning signal. *(c) MCTS Round 10*: The policy stabilizes, maintaining cluster diversity while selecting representative samples from each region. This exploration-to-exploitation transition emerges naturally from UCB-guided search, adaptively focusing on trajectory combinations where the model benefits most.

Theoretical Analysis. We provide theoretical guarantees for our MCTS-based learning approach, demonstrating its convergence to a near-optimal traversal policy with polynomial sample complexity. The analysis proceeds by showing that the key components of the learned policy converge to their optimal counterparts. Specifically, we establish convergence for: (1) the discrete choices at action nodes, and (2) the continuous allocation ratios at expansion nodes.

(1) Action Node Convergence. An action node $\hat{v}$ must learn the optimal binary action $a^* \in \{0, 1\}$, which corresponds to either terminating the search or expanding to child nodes. Let $\mu_a^* = \mathbb{E}[R_a]$ be the true expected reward for taking action a. The optimal action is then defined as $a^* = \arg\max_{a \in \{0,1\}} \mu_a^*$. Our algorithm estimates this true expected reward using the empirical mean, $\hat{\mu}_a(n_a) = \frac{1}{n_a} \sum_{t=1}^{n_a} R_a(t)$, which is calculated after action a has been selected n_a times. The algorithm's decision rule is to select the action with the highest empirical reward, i.e., $a_{\text{selected}} = \arg\max_{a \in \{0,1\}} \hat{\mu}_a(n_a)$. The following theorem, which leverages standard concentration inequalities, establishes that with sufficient visits, the empirically optimal action converges to the true optimal action.

Theorem 3 (Action Node Convergence). *Let $\hat{v}$ be an action node, and assume one action is strictly superior, i.e., there exists a gap $\Delta > 0$ such that $\mu_{a^*}^* \geq \mu_{a'}^* + \Delta$ for the sub-optimal action a'. Let all rewards be bounded within $[0, R_{\max}]$. For any desired confidence level $\delta \in (0, 1)$, if each action $a \in \{0, 1\}$ has been sampled at least $n_a \geq \frac{8R_{\max}^2 \ln(4/\delta)}{\Delta^2}$ times, then with a probability of at least $1 - \delta$, the empirically selected action is the true optimal one:* $\arg\max_{a \in \{0,1\}} \hat{\mu}_a(n_a) = a^*$.

Therefore, the action node converges to the optimal action with probability $1 - \delta$ after collecting $O(R_{\max}^2 \ln(1/\delta)/\Delta^2)$ samples. The detailed proof is available in the appendix [52].

(2) Expansion Node Convergence. We establish the convergence guarantee for allocation ratios at each expansion node.

Theorem 4 (Allocation Ratio Convergence). *Consider an expansion node $\bar{v}$ with k children. Let $\hat{\mu}_i(n_i)$ denote the empirical mean reward for child i after n_i visits, and let μ_i^* denote the true expected reward. Define the allocation ratios as: $r_i(n) = \frac{\hat{\mu}_i(n_i)}{\sum_{j=1}^k \hat{\mu}_j(n_j)}$, $r_i^* = \frac{\mu_i^*}{\sum_{j=1}^k \mu_j^*}$. We can assume that: (1) Rewards are bounded: $R_i \in [0, R_{\max}]$ for all i; (2) $\mu^*_{\min} := \min_{i=1,\dots,k} \mu_i^* > 0$; (3)$\mu^*_{sum} := \sum_{j=1}^k \mu_j^* > 0$. Then, for any $\epsilon \in (0,1)$ and $\delta \in (0,1)$, if each child is visited at least $n_{\min} = \frac{8k^2 R_{\max}^2 \ln(2k/\delta)}{\epsilon^2 (\mu^*_{sum})^2}$ times, then $\mathbb{P}\left(\max_{i=1,\dots,k} |r_i(n) - r_i^*| \le \epsilon\right) \ge 1 - \delta$.*

Therefore, the expansion node converges to the optimal allocation ratio with probability $1-\delta$ by sampling $n_{\min} = O\left(\frac{k^2 R_{\max}^2 \ln(k/\delta)}{\epsilon^2 (\mu^*_{\text{sum}})^2}\right)$ times for each child. The detailed proof is available in [52].

6 Weighted Learning

Motivation. While our MCTS-based sampling strategy generates trajectory subsets $\mathcal{T}$ that approximate the optimal trajectory distribution $P(\mathcal{T}) \approx P(\mathcal{T}^*)$, a critical challenge remains: the distribution of constructed training pairs may not align with the optimal training distribution. Specifically, even if we sample trajectories optimally, the resulting trajectory pairs $\mathcal{TP}_{\text{train}}$ constructed from pairwise combinations may satisfy $P(\mathcal{TP}_{\text{train}}) \neq P(\mathcal{TP}^*_{\text{train}})$ due to the quadratic explosion in pair space. This distribution mismatch creates two fundamental problems. First, since our sampling focuses on individual trajectory selection, it cannot directly control the distribution of trajectory pairs, which have their own similarity characteristics. Second, each round only selects a subset of trajectories, resulting in training data that captures only a partial view of the global trajectory similarity landscape. These issues can lead to biased model learning where certain types of trajectory relationships are over- or under-represented. To address these challenges, we propose a weighted learning approach that adaptively reweights training examples to better approximate the optimal training distribution while leveraging cached data from previous rounds to improve distribution coverage.

Reweighting Framework. Standard training minimizes the uniform loss across all training pairs: $\mathcal{L}_{std} = \frac{1}{|\mathcal{TP}_{\text{train}}|} \sum_{(i,j)} \ell_{i,j}(\theta)$, where $\ell_{i,j}(\theta) = \text{loss}(g_\theta(T^i, T^j), f^{i,j})$ represents the prediction loss for trajectory pair (T^i, T^j) with ground truth similarity $f^{i,j}$. We reformulate this as a weighted optimization problem: $\mathcal{L}_{weighted} = \sum_{(i,j)} w_{i,j} \ell_{i,j}(\theta)$, where $w_{i,j}$ represents the importance weight for each training pair. However, learning individual pair weights is computationally expensive-with N trajectories, we would need to optimize $O(N^2)$ weight parameters. To achieve tractability, we propose trajectory-level weighting where each trajectory T^i receives a weight w_i, and pair weights are derived as $w_{i,j} = \frac{w_i + w_j}{2}$. This reduces the parameter space from $O(N^2)$ to $O(N)$ while maintaining the ability to emphasize important trajectories across all their pairwise interactions. The bilevel optimization problem becomes:

$$w^* = \arg\min_{w \ge 0} \mathcal{L}_{\text{val}}(\theta^*(w)), \quad \theta^*(w) = \arg\min_{\theta} \sum_{(i,j)} \frac{w_i + w_j}{2} \ell_{i,j}(\theta) \tag{6}$$

where $\mathcal{L}_{\text{val}}(\theta) = \frac{1}{|\mathcal{TP}_{\text{val}}|} \sum_{(i,j) \in \mathcal{TP}_{\text{val}}} \ell_{i,j}(\theta)$ evaluates performance on validation data, and $w \ge 0$ ensures training stability.

Cache-Enhanced Validation. A critical component of our approach is constructing representative validation sets for weight optimization. Using only current-round data would suffer from the same distribution bias we aim to correct. Instead, we maintain a cache $\mathcal{TP}$ that accumulates trajectory pairs from previous rounds, providing broader coverage of the trajectory similarity landscape. The cache is initialized with randomly sampled pairs from the global trajectory pool and continuously updated: $\mathcal{TP} \leftarrow \mathcal{TP} \cup \mathcal{TP}^i_{\text{train}}$ after each round i. This design offers two key advantages: (1)

Algorithm 1: Cache-enhanced Weighted Learning

Input: Cache $\mathcal{TP}$, model g_θ, training data $\mathcal{TP}_{\text{train}}$, trajectories $\mathcal{T}$, learning rate α and β, iterations I_t, batch sizes bs_1, bs_2

1 Initialize weights: $w_i \leftarrow \frac{1}{|\mathcal{T}|}$ for each $T^i \in \mathcal{T}$;
2 Update cache: $\mathcal{TP} \leftarrow \mathcal{TP} \cup \mathcal{TP}_{\text{train}}$;
3 **for** $t = 1 \rightarrow I_t$ **do**
4 Sample training batch: $\mathcal{TP}^t_{\text{train}} \leftarrow \text{SampleBatch}(\mathcal{TP}_{\text{train}}, bs_1)$;
5 Sample validation batch: $\mathcal{TP}^t_{\text{val}} \leftarrow \text{SampleBatch}(\mathcal{TP}, bs_2)$;
/* Virtual model update with current weights */
6 Compute weighted loss: $\mathcal{L}_{\text{train}} = \sum_{(i,j)\in\mathcal{TP}^t_{\text{train}}} \frac{w_i+w_j}{2} \ell_{i,j}(\theta_t)$;
7 Virtual update: $\tilde{\theta}_t \leftarrow \theta_t - \alpha\nabla_\theta \mathcal{L}_{\text{train}}$;
/* Update weights based on validation loss */
8 Compute validation loss: $\mathcal{L}_{\text{val}} = \frac{1}{m} \sum_{(i,j)\in\mathcal{TP}^t_{\text{val}}} \ell_{i,j}(\tilde{\theta}_t)$;
9 Weight gradients: $\nabla w \leftarrow \nabla_w \mathcal{L}_{\text{val}}$;
10 Project and normalize: $w \leftarrow \text{normalize}(\max(w - \beta\nabla w, 0))$;
/* Actual model update with updated weights */
11 Recompute loss: $\mathcal{L}_{\text{train}} = \sum_{(i,j)\in\mathcal{TP}^t_{\text{train}}} \frac{w_i+w_j}{2} \ell_{i,j}(\theta_t)$;
12 Update model: $\theta_{t+1} \leftarrow \theta_t - \alpha\nabla_\theta \mathcal{L}_{\text{train}}$;

the accumulated data better approximates the global distribution $P(\mathcal{TP}^*_{\text{train}})$ as more rounds are included, and (2) we avoid recomputing expensive ground truth similarities for cached pairs. For validation, we sample $\mathcal{TP}_{\text{val}} \sim \text{sample}(\mathcal{TP})$ from the cache, ensuring our weight learning process optimizes against a more representative distribution than current-round data alone.

Online Optimization Algorithm. Solving the bilevel optimization problem (Equation 6) exactly requires nested loops of optimization, which is computationally prohibitive. We adopt an online approximation strategy that alternately updates weights and model parameters in a single optimization step per iteration. Algorithm 1 presents our complete procedure. We now walk through a single iteration to explain how each step contributes to solving the bilevel optimization:

Step 1: Batch Sampling (Lines 4-5). We sample a training batch $\mathcal{TP}^t_{\text{train}}$ from current-round data and a validation batch $\mathcal{TP}^t_{\text{val}}$ from the cache. The cached validation set provides broader distribution coverage for weight optimization.

Step 2: Virtual Model Update (Lines 6-7). We compute the weighted training loss and perform a virtual (temporary) gradient descent step on model parameters: $\tilde{\theta}_t \leftarrow \theta_t - \alpha\nabla_\theta \mathcal{L}_{\text{train}}$. This step approximates the inner-level optimization $\theta^*(w)$ in Equation 6, simulating how the model would change under current weights.

Step 3: Weight Update (Lines 8-10). Using the virtually updated model $\tilde{\theta}_t$, we evaluate validation loss and compute weight gradients via backpropagation through the virtual update. The weights are then updated by gradient descent with projection onto the non-negative simplex. This step approximates the outer-level optimization, adjusting weights to minimize validation loss.

Step 4: Actual Model Update (Lines 11-12). With the updated weights, we recompute the weighted training loss and perform the actual model parameter update. This completes one iteration of the bilevel optimization.

The algorithm's effectiveness stems from its ability to identify trajectories that, when emphasized during training, lead to better validation performance on the broader cached distribution. This creates an implicit mechanism for correcting the distribution mismatch between the sampled training data and the optimal global distribution.

Table 2. Statistics of Datasets

Dataset	Time Span	Cover Area (km^2)	#traj.	#points /traj.	Avg. length (m)
Porto	a year	16.0 × 20.1	1,259,639	50	6,445
Chengdu	a week	9.7 × 9.7	1,380,777	142	4,586
Germany	-	1,023 × 1,187	243,417	84	338,622

Convergence Analysis. The problem of learning optimal trajectory weights is formulated as a bilevel optimization program:

$$w^* = \arg\min_{w\in\Delta} F(w) \triangleq \mathcal{L}_{\text{val}}(\theta^*(w)) \quad \text{s.t.}\theta^*(w) = \arg\min_{\theta} \mathcal{L}_{\text{train}}(\theta, w)$$

We prove that our online, single-loop algorithm converges to a stationary point of the outer objective $F(w)$. The core of our analysis involves bounding the error between the true (but infeasible) bilevel gradient $\nabla F(w_t)$ and our algorithm's efficient, single-step approximation. We show this error is controlled by the inner-loop learning rate and the suboptimality of the model parameters. Leveraging this bound within a descent analysis demonstrates that our weight updates ensure a sufficient decrease in the outer objective. This guarantees that the algorithm converges to a neighborhood of a stationary point, with the average squared gradient norm satisfying: $\lim_{I_t\to\infty} \frac{1}{I_t} \sum_{t=1}^{I_t-1} \mathbb{E}[\|\nabla F(w_t)\|^2] = O(\alpha^2) + O(\beta)$. This result establishes the algorithm's stability and convergence. The complete theoretical analysis and detailed proofs are provided in [52].

7 Experiment

7.1 Experimental Setup

Datasets. As shown in Table 2, following [9], we conduct a series of experiments on three real-world datasets.

Similarity Learning Models. To establish a comprehensive and fair evaluation framework, we select representative state-of-the-art models from four major categories: (1) RNN-based models: TMN [45]; (2) Self-attention based models: T3S [46] (which incorporates both RNN and self-attention mechanisms); (3) CNN-based models: TrjSR [6]; (4) GNN-based models: TrajGAT [49].

Baseline Methods. We compare against the following methods:

• **Rand** [9]: The random selection method is widely used for preparing training data in existing similarity learning papers, making it a natural baseline. We extend this approach to iteratively sample trajectories across multiple training rounds.

• **Coreset** [7]: Since direct gradient computation for coreset selection is not feasible in our setting, we adapt a cluster-based coreset selection method. This approach first clusters trajectories, then uses the cluster distribution of trajectories to guide the coreset sampling of training data for each round.

• **Active** [32]: This state-of-the-art active learning method for data augmentation iteratively selects high-quality data based on predefined metrics between training and validation sets. We adapt this method to iteratively sample trajectories based on their feature distribution similarity with the validation data.

• **Curric** [61]: This state-of-the-art curriculum learning method for individual trajectory learning determines the learning order based on curriculum difficulty scores. We extend this approach by ranking all trajectories according to the same difficulty metric, then iteratively sampling a fixed number of trajectories to construct training data.

Similarity Labels & Metrics. Following established benchmarking practices [9], we employ four widely-used distance functions as ground truth for trajectory similarity: DTW [51], ERP [10], Fréchet [16], and Hausdorff [2]. For evaluation, we use a test set of 2000 trajectories. Each similarity learning model generates trajectory embeddings, and for each query trajectory, we compute pairwise

Table 3. Performance comparison on Porto, Chengdu and Germany datasets. **Bold** and underlined values indicate the best and second-best results, respectively. Each cell format: ①/②/③/④ represents TMN/T3S/TrjSR/TrajGAT performance. Similarity measures: Ⓓ=DTW, Ⓔ=ERP, Ⓕ=Fréchet, Ⓗ=Hausdorff. Methods: A=**Rand**, B=**Coreset**, C=**Active**, D=**Curric**, E=**GoodTP**.

		Porto			Chengdu			Germany		
Sim		HR@1(%)	HR@10(%)	HR@50(%)	HR@1(%)	HR@10(%)	HR@50(%)	HR@1(%)	HR@10(%)	HR@50(%)
Ⓓ	A	34.9/20.7/22.5/25.3	55.8/40.2/31.8/38.5	72.8/56.8/38.5/49.7	28.2/16.1/33.6/9.5	44.6/29.6/41.8/26.8	57.4/46.5/48.9/43.5	34.9/32.5/34.4/14.6	52.7/51.0/52.0/16.6	67.3/66.6/54.3/29.7
	B	37.7/21.8/28.9/25.8	58.9/40.1/37.1/39.9	75.2/57.7/42.9/51.4	29.5/16.9/34.7/13.7	46.3/33.7/42.6/29.3	59.0/44.5/49.9/47.5	30.0/31.7/27.3/**15.5**	48.8/50.4/47.3/**17.1**	61.4/65.7/59.5/30.6
	C	35.7/22.5/24.6/25.8	58.0/41.8/33.1/39.7	74.4/57.4/39.8/51.8	27.7/18.3/35.0/13.5	45.0/33.7/42.4/30.1	57.6/47.3/49.6/47.7	**37.3**/31.0/33.9/15.3	52.3/50.5/58.0/16.9	**67.7**/66.0/67.1/30.2
	D	36.6/23.4/36.5/**25.9**	57.2/41.6/42.7/**40.8**	73.7/58.2/44.5/**54.1**	29.1/17.1/35.5/10.7	45.1/20.7/42.7/23.9	57.4/22.9/49.9/42.4	34.5/17.5/34.1/14.1	52.4/29.5/53.8/16.1	66.6/50.5/61.1/28.8
	E	**40.3**/**27.9**/**39.4**/25.2	**62.6**/**45.3**/**50.4**/36.7	**77.4**/**61.1**/**51.2**/49.2	**29.6**/**19.5**/**39.3**/**20.0**	**47.9**/**34.3**/**48.5**/**39.5**	**60.2**/**49.4**/**54.5**/**57.1**	34.3/**34.5**/**35.5**/15.2	**52.9**/**54.0**/**58.4**/**17.1**	67.1/**67.8**/**67.2**/**32.3**
Ⓔ	A	39.2/27.7/23.4/11.3	66.3/52.6/45.1/22.1	92.9/86.9/59.3/30.2	51.4/57.2/23.1/7.9	70.2/85.5/34.0/24.2	95.1/96.0/60.4/38.7	77.8/57.8/18.9/9.6	78.2/73.1/31.3/18.0	87.4/84.8/43.9/27.0
	B	39.9/23.3/24.2/11.6	67.1/53.0/44.4/22.6	92.7/82.4/59.3/30.7	51.2/54.8/23.9/9.9	72.7/88.4/33.4/27.2	**95.6**/90.0/61.9/43.9	71.8/40.1/18.7/9.8	73.0/59.3/31.6/18.3	79.3/73.2/44.3/27.6
	C	46.4/25.7/23.2/11.5	68.0/55.7/43.5/22.3	92.2/87.8/56.8/30.3	**51.8**/54.2/23.2/**10.5**	70.8/88.1/37.6/**29.6**	95.3/88.5/63.6/**46.3**	74.6/62.5/18.0/**10.2**	75.2/65.3/31.8/18.4	83.2/74.1/**44.8**/28.3
	D	33.2/24.4/**24.5**/11.4	49.6/50.9/43.5/22.7	61.2/85.4/56.1/31.1	38.7/53.3/24.4/9.8	54.2/87.6/37.5/27.4	92.7/89.8/61.9/41.4	70.4/65.7/**19.0**/10.0	72.9/83.3/31.1/18.2	85.6/91.4/43.7/27.8
	E	**57.2**/**46.0**/23.9/**15.3**	**78.3**/**73.5**/**45.7**/**32.8**	**93.0**/**90.7**/**59.4**/**35.0**	50.8/**61.8**/**28.1**/10.2	**72.9**/**89.6**/**40.5**/29.0	95.5/**96.6**/**65.9**/40.9	**79.5**/**84.8**/18.5/10.1	**80.6**/**88.4**/**32.0**/**18.6**	**88.5**/**92.4**/44.5/**28.8**
Ⓕ	A	44.4/38.8/34.7/20.6	63.6/59.9/**45.4**/32.1	77.2/74.7/49.6/42.7	59.9/27.1/31.9/18.8	76.5/46.8/44.9/41.5	82.8/61.6/47.5/58.5	51.4/52.4/32.5/14.5	69.3/77.7/63.9/16.5	80.8/90.0/**81.5**/30.6
	B	43.6/40.7/26.8/24.2	63.6/59.3/36.3/42.7	76.5/74.2/46.5/**58.6**	**60.6**/25.5/34.4/20.5	76.9/45.9/46.9/44.2	82.4/60.5/55.8/60.2	57.9/42.0/30.0/14.3	76.5/76.4/55.8/17.1	87.8/85.3/65.2/32.4
	C	45.7/40.5/30.6/24.8	64.1/60.4/40.4/42.6	77.3/74.6/49.7/58.4	59.1/23.0/35.9/14.5	76.5/47.4/48.8/35.9	82.6/63.1/60.2/53.9	57.8/42.4/29.7/14.0	76.4/76.5/55.6/16.9	87.8/85.3/64.3/31.9
	D	47.0/41.0/35.1/20.5	66.4/60.4/42.3/33.4	78.5/74.8/49.1/43.8	59.5/26.3/34.7/21.2	76.3/44.1/47.7/43.9	81.8/56.1/58.8/59.6	58.9/33.5/30.5/14.1	78.2/56.3/58.3/17.1	89.2/77.3/77.4/32.0
	E	**50.6**/**44.6**/**35.5**/**28.1**	**67.0**/**62.9**/44.5/**43.2**	**79.2**/**76.3**/**49.8**/56.0	60.0/**27.3**/**38.8**/**22.2**	**77.5**/**48.4**/**50.7**/**45.5**	**83.0**/**63.8**/**68.2**/**61.2**	**60.9**/**59.6**/**35.1**/**14.9**	**79.3**/**80.9**/**65.0**/**17.8**	**90.6**/**91.1**/81.1/**32.8**
Ⓗ	A	37.1/5.9/45.9/16.6	57.4/16.6/60.4/26.5	73.8/34.8/72.0/34.6	42.0/14.6/41.8/10.4	67.1/32.5/60.5/22.4	81.4/52.7/73.9/40.2	52.1/28.7/44.0/14.7	75.8/52.5/66.3/16.0	88.5/75.4/82.1/30.6
	B	38.7/5.3/44.8/17.9	57.4/15.3/59.8/29.3	73.6/31.8/70.4/39.5	45.3/16.6/44.2/15.3	69.1/35.6/64.1/35.6	82.1/56.7/76.7/57.5	47.8/31.2/45.0/17.0	74.3/60.4/68.1/17.4	88.2/82.2/81.0/31.8
	C	39.9/6.6/45.7/17.9	58.2/18.1/59.5/28.1	74.1/36.2/70.9/37.2	46.2/18.9/43.9/12.6	69.6/38.9/56.7/32.0	82.2/60.9/58.8/55.0	48.8/16.5/43.1/17.5	75.2/48.3/61.6/18.6	88.5/66.2/67.4/**33.3**
	D	42.3/8.2/46.4/17.1	61.1/24.0/60.3/26.8	75.8/43.6/70.3/34.7	46.3/21.0/42.9/13.2	70.4/35.8/56.6/32.9	81.1/57.5/65.9/55.7	53.0/19.4/41.8/17.8	**78.2**/49.6/63.1/18.3	91.1/66.8/77.4/32.9
	E	**45.7**/**23.8**/**46.6**/**26.5**	**64.3**/**42.5**/**61.1**/**40.0**	**76.9**/**60.7**/**72.1**/**51.3**	**49.3**/**37.6**/**48.0**/**18.8**	**71.9**/**56.5**/**67.7**/**39.3**	**83.2**/**76.3**/**78.1**/**59.7**	**55.7**/**45.4**/**49.0**/**18.1**	77.9/**73.3**/**76.9**/**22.1**	**91.5**/**88.3**/**89.7**/32.8

Remark. Due to space constraints, we report HR@1, HR@10, and HR@50. Complete results including HR@5 are provided in the appendix [52].

Euclidean distances in the embedding space to retrieve the top-α most similar trajectories. We assess model performance using the hit rate metric $HR@\alpha$ [19, 48, 49], where higher values indicate more accurate trajectory similarity computation. Specifically, for each query trajectory, $HR@\alpha$ calculates the proportion of true top-α neighbors present in the retrieved set: $HR@\alpha = \frac{\sum_{i=1}^{N} |\text{Top-}\alpha_{\text{emb}}^{(i)} \cap \text{Top-}\alpha_{\text{gt}}^{(i)}|}{N\alpha}$, where N is the number of query trajectories, and Top-$\alpha_{\text{emb}}^{(i)}$ and Top-$\alpha_{\text{gt}}^{(i)}$ represent the top-α most similar trajectories to the i-th query trajectory according to the embedding distance and ground truth distance, respectively. We evaluate performance at different Top-α settings (i.e., $\alpha \in \{1, 5, 10, 50\}$) to capture both precise and broader similarity matching capabilities.

Experimental Settings. All experiments were conducted using PyTorch 2.1.0 and Python 3.9.12 on an RTX 4090 GPU running Ubuntu 24.04.1. We employed the Adam optimizer [25] with a learning rate of 0.001 and trained all models for up to 20 epochs with early stopping based on validation performance. The default batch size was set to 256, with adjustments made when GPU memory constraints required (e.g., TrajGAT used a batch size of 64 due to higher memory requirements). Following established practices [9], we set the embedding dimensionality to 128 for all models. For spatial discretization in grid-based methods, we used 100-meter grid cells. Model-specific configurations followed their original papers: TrjSR used 162×128 image resolution with 10 CNN layers, while other models employed 2 encoder layers by default. Since our method focuses on adjusting trajectory selection in training data, we maintained identical training conditions across all baseline comparisons to ensure fairness. For example, the sampling budget B was set to 1000k trajectory pairs for all methods.

7.2 Effectiveness Evaluation

As shown in Table 3, the experimental results demonstrate several key insights regarding the effectiveness of different trajectory selection methods across multiple similarity metrics and datasets.

(1) Overall Superiority of Our Method: Our method consistently outperforms all baseline methods across the majority of experimental configurations. Specifically, **GoodTP** achieves the best performance in around 90% of metric-model combinations across all three datasets, demonstrating its robustness across different geographical contexts and trajectory characteristics. Therefore,

Table 4. Ablation studies. Each cell format: ①/② represents T3S/TrajGAT performance. Ⓓ=DTW, Ⓗ=Hausdorff.

Sim	methods	Porto			Chengdu		
		HR@1(%)	HR@10(%)	HR@50(%)	HR@1(%)	HR@10(%)	HR@50(%)
Ⓓ	w/o SD	26.0/24.9	45.3/34.5	58.9/43.5	19.3/18.9	34.8/37.9	48.5/53.2
	w/o MP	24.7/24.2	45.3/35.5	59.5/48.1	17.2/23.3	31.1/36.7	48.6/48.6
	w/o Clust	19.9/20.8	40.8/31.8	59.2/40.8	18.9/15.0	28.4/25.5	41.3/33.3
	w/o MCTS	23.3/12.9	43.0/26.8	60.7/41.7	10.7/18.4	25.1/30.3	37.4/37.2
	w/o weight	27.0/25.8	44.6/36.6	61.1/47.8	19.5/15.5	33.6/33.4	47.1/49.9
	DBSCAN	27.4/19.0	40.8/31.6	58.8/47.7	17.3/16.5	30.2/34.0	45.6/48.8
	GoodTP	**27.9/25.2**	**45.3/36.7**	**61.1/49.2**	**19.5/20.0**	**34.3/39.5**	**49.4/57.1**
Ⓗ	w/o SD	18.9/26.8	31.9/36.8	55.4/41.6	39.6/17.1	54.5/39.3	76.3/58.2
	w/o MP	20.7/26.2	34.3/34.7	55.4/41.0	34.7/18.8	54.0/38.5	74.0/55.9
	w/o Clust	8.8/10.8	24.6/20.9	34.0/30.1	13.4/11.7	32.7/23.6	51.7/34.8
	w/o MCTS	7.4/12.3	23.0/21.2	31.7/30.3	18.0/13.1	37.2/23.5	59.0/36.5
	w/o weight	20.4/22.6	39.5/37.4	58.2/49.4	35.7/17.9	55.2/38.7	74.7/58.9
	DBSCAN	11.4/21.8	26.3/37.3	51.2/51.3	20.4/18.6	41.3/38.4	63.5/58.6
	GoodTP	**23.8/26.5**	**42.5/40.0**	**60.7/51.3**	**37.6/18.8**	**56.5/39.3**	**76.3/59.7**

our method is data-agnostic and can be applied across datasets without dataset-specific feature engineering.

(2) Similarity Metric Impact: Performance gains vary significantly across metrics: *(i) DTW:* Consistent improvements across models, with notable gains for TMN and T3S on Porto but more modest improvements on Chengdu, suggesting dataset-specific influences. *(ii) ERP:* Most dramatic improvements, particularly for TMN and T3S on Porto , indicating high sensitivity to training data quality. *(iii) Fréchet:* Moderate but consistent 10-20% relative gains across models, suggesting effective trajectory diversity benefits. *(iv) Hausdorff:* Strong improvements for memory-intensive models like TMN and T3S , indicating the importance of representative trajectory pattern selection.

(3) Model-Specific Analysis: Different models exhibit distinct response patterns: *(i) TMN* shows the most substantial improvements across all distances (average 25% relative gain), indicating RNN-based architectures benefit significantly from quality training data. *(ii) T3S* demonstrates strong improvements under ERP and Hausdorff distances, with self-attention architecture showing high sensitivity to selection quality. *(iii) TrjSR* exhibits moderate improvements across configurations. As an image-based approach, it benefits from diverse spatial patterns while being less sensitive to individual trajectory selection. *(iv) TrajGAT* shows variable performance across distances with generally smaller improvements, possibly due to graph attention mechanisms handling noisy data more robustly.

(4) Baseline Method Comparison: Among the baseline methods, **Coreset** and **Curric** generally perform better than **Rand** and **Active**, but still fall significantly short of our method. For example, under DTW similarity on Porto, **Coreset** achieves 37.7% HR@1 for TMN compared to our 40.3%, while **Curric** reaches 36.6%. This indicates that while sophisticated selection strategies provide improvements over random sampling, our method captures trajectory characteristics more effectively than existing approaches.

(5) Scalability with Ranking Metrics: The performance improvements are consistent across different $HR@\alpha$ values, with particularly strong gains at higher α values. This suggests that our method not only improves the top-1 accuracy but also enhances the overall ranking quality, which is crucial for practical trajectory similarity applications where users may be interested in multiple similar trajectories rather than just the most similar one.

Table 5. Efficiency analysis results. Each cell format: ①/② represents T3S/TrajGAT. Ⓓ=DTW, Ⓗ=Hausdorff.

Sim	Methods	Porto			Chengdu		
		Traj Sel (s)	Data Constr (s)	Model Train (s)	Traj Sel (s)	Data Constr (s)	Model Train (s)
Ⓓ	**Rand**	0.02/0.02	95.15/66.38	249.28/1670.18	0.003/0.004	387.79/201.68	331.53/5638.98
	Coreset	54.3/50.40	121.69/84.85	220.47/2311.28	12.61/8.51	423.86/193.47	305.98/6901.15
	Active	56.71/49.94	91.87/68.16	240.29/1829.03	11.67/10.38	350.17/160.18	304.86/5432.57
	Curric	4.17/7.40	100.95/78.76	178.39/1541.99	7.76/8.12	466.04/191.75	203.71/4084.53
	GoodTP	57.22/60.82	12.14/36.08	765.91/8047.25	27.9/33.83	30.81/52.06	907.61/20051.26
	w/o weight	57.37/37.74	5.77/37.33	270.27/2215.53	29.75/24.31	28.79/68.49	426.03/6493.48
Ⓗ	**Rand**	0.02/0.02	53.41/74.43	256.66/1915.63	0.002/0.002	137.34/118.83	320.51/5507.15
	Coreset	56.37/61.60	64.97/68.60	263.38/2311.61	9.47/8.18	154.45/110.38	314.38/6670.88
	Active	55.75/61.21	51.3/59.59	139.91/1805.47	9.68/8.43	124.1/92.48	312.76/5212.92
	Curric	4.32/4.55	54.9/58.22	177.05/1624.61	8.03/8.29	156.1/102.15	206.16/4116.64
	GoodTP	54.85/47.66	10.03/46.81	772.4/7982.06	28.72/20.80	17.65/47.62	918.38/18396.44
	w/o weight	60.61/37.17	6.32/37.22	285.74/2198.40	30.41/20.92	14.8/49.61	326.66/5342.52

7.3 Ablation Study

In this section, we evaluate each component through ablation tests. For trajectory feature extraction, we incorporate each feature with w/o SD (spatial distribution features) and w/o MV (movement pattern features). For clustering, we replace our clustering with the SOTA deep learning-based method [18], denoted as w/o Clust. For MCTS, we replace the MCTS policy with greedy policy at each node, denoted as w/o MCTS. Finally, we denote w/o weight by replacing the weighted learning component. Additionally, to assess how different clustering backbones affect downstream performance, we compare our default K-means-based hierarchy against a DBSCAN-based [17] variant, keeping all other components (MCTS selection and weighted learning) fixed. As shown in Table 4, we have the following observations:

(1) Feature Components Show Variable Importance. The ablation results reveal that different features contribute distinctly to performance. SD removal results in comparable drops (1-2% on Porto, 0-2% on Chengdu), indicating spatial distribution features offer incremental gains. However, removing MP leads to more substantial performance loss (2-3% on Porto, 1-3% on Chengdu), demonstrating that movement pattern features are more critical for capturing trajectory dynamics.

(2) Clustering and MCTS Show Similarity-Dependent Criticality. For DTW similarity, the clustering component proves most critical—its removal causes severe degradation (7-8% drop on Porto for T3S, 9-11% on Chengdu). This suggests DTW-based methods benefit heavily from organized trajectory grouping that exploits temporal alignment patterns. Conversely, for Hausdorff distance, the MCTS component is most vital, with removals causing dramatic performance drops (15-16% on Porto for T3S, 18-20% on Chengdu), indicating that strategic search is essential for optimizing discrete point-wise distance metrics.

(3) Dataset Complexity Amplifies Component Sensitivity. Chengdu consistently shows greater sensitivity to component removal than Porto, particularly for clustering (11% vs. 8% drop for DTW) and MCTS (20% vs. 16% drop for Hausdorff). This likely reflects Chengdu's more complex features, where strategic exploration and organized grouping become substantially more critical for maintaining high-quality trajectory selection performance.

(4) Clustering Algorithm Choice Affects Performance. GoodTP with K-means consistently outperforms the DBSCAN variant, which still substantially exceeds the "w/o Clust" baseline. For DTW, DBSCAN shows minimal degradation (Porto-T3S: 27.4% vs. 27.9% HR@1); for Hausdorff, the gap is larger (Porto-T3S: 11.4% vs. 23.8%) yet DBSCAN still outperforms "w/o Clust" (11.4% vs. 8.8%). This confirms that K-means produces more balanced trees for effective MCTS exploration.

7.4 Efficiency Evaluation

To evaluate the computational efficiency of our proposed **GoodTP** method, we conduct comprehensive experiments measuring the time breakdown across three critical phases: *trajectory selection*, *data construction*, and *model training*. We compare **GoodTP** against four baseline methods across two datasets, two distance functions (DTW and Hausdorff), and two learning models (T3S and TrajGAT). Additionally, we include a variant of our method without the weighting mechanism (w/o weight) to isolate the computational overhead of this component. The results are shown in Table 5, and we have the following observations:

(1) Training Time Dominance and Model Architecture Impact. Model training constitutes 60-85% of total execution time due to intensive forward passes and gradient computation during backpropagation. TrajGAT requires substantially longer training times compared to T3S because TrajGAT employs graph neural networks with complex message passing over trajectory structures, while T3S uses more efficient attention mechanisms. Dataset characteristics also significantly impact training duration.

(2) Trajectory Selection Efficiency: Algorithmic Trade-offs. Rand and **Curric** demonstrate superior selection efficiency through straightforward random sampling and simple sorting operations. **Active** and **Coreset** require more time due to the iterative optimization and the clustering process. Our **GoodTP** method maintains competitive efficiency using MCTS, which achieves high efficiency through strategic search space pruning. Importantly, selection overhead represents only 2-8% of total execution time, making this computational investment acceptable given the trajectory sampling quality improvements.

(3) Data Construction Revolution Through Caching Mechanisms. GoodTP achieves remarkable efficiency gains in data construction, reducing processing time by 70-90% compared to baseline methods. This dramatic improvement results from our novel caching mechanism that eliminates redundant ground truth computations. Conventional methods repeatedly calculate distance metrics and similarity measures for overlapping trajectory segments across different training iterations, leading to substantial computational waste.

(4) Training Overhead: A Deliberate Trade-off for Adaptive Learning. GoodTP incurs extended training time due to its online learning mechanism, which requires continuous validation to adaptively adjust sample importance. This process introduces additional forward passes and gradient computations beyond standard training loops, effectively increasing computational graph complexity. However, our ablation study with the w/o weight variant provides crucial evidence: removing the weighting mechanism reduces training time to competitive levels. This confirms that the training overhead is not due to algorithmic inefficiency, but rather a *deliberate design choice* that trades computational cost for superior model performance. Importantly, this trade-off is *tunable*. Practitioners can balance training effectiveness and efficiency by applying the adaptive mechanism at regular intervals (e.g., every k iterations) rather than at every iteration.

These results demonstrate that **GoodTP** achieves a strategic efficiency optimization by investing computational resources in the learning process (adaptive weighting) while dramatically reducing costs in the preparation process (data selection and pair construction). This trade-off is acceptable because it establishes a favorable paradigm: **invest computational resources during training (a one-time cost) to achieve superior model quality and faster inference (continuous benefits in production deployment)**.

7.5 Evaluation on Hyperparameters

We evaluate three key hyperparameters: cluster number K, iteration number I_t, and training budget B. Following [3], we recommend a logarithmic (geometric-progression) search strategy for all three

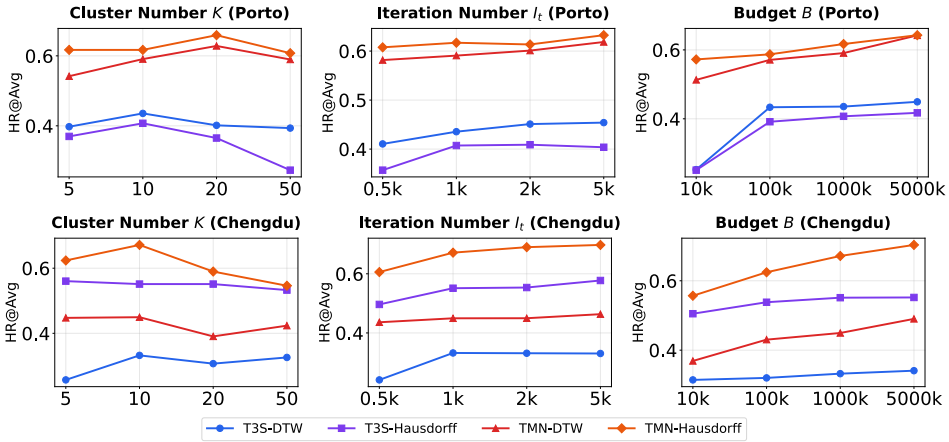


Fig. 5. Effectiveness Evaluation on Hyperparameters

hyperparameters, as sensitivity often scales approximately logarithmically. We report the average across HR@{1,5,10,50} for two models (T3S, TMN) with two distance functions (DTW, Hausdorff) on Porto and Chengdu datasets. As shown in Fig. 5, we have the following observations and practical tuning guidelines.

(1) Cluster Number K: Cluster granularity impact varies significantly across models and datasets. On Porto, TMN achieves optimal performance at $K = 20$, while T3S peaks at $K = 10$, suggesting self-attention architectures benefit from coarser clustering. On Chengdu, TMN-Hausdorff degrades significantly from $K = 10$ to $K = 50$, indicating excessive fragmentation disrupts selection. The results suggest $K = 10 \sim 20$ provides optimal balance between flexibility and pattern coherence.

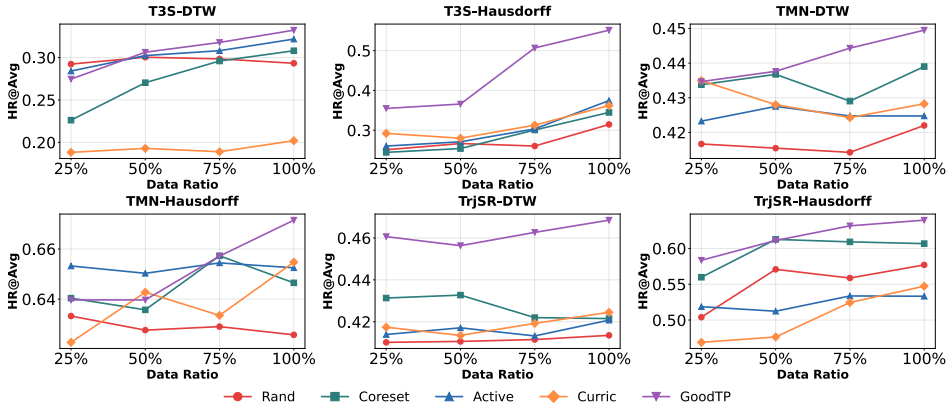
Tuning guideline. In our MCTS formulation, K controls the search tree's branching factor. A small initial K keeps the action space manageable, allowing the UCB policy to converge reliably during early tuning. Therefore, for new datasets, we recommend starting with a small K and increasing it geometrically (e.g., $5, 10, \ldots$). The search stops once validation gains drop below a marginal threshold.

(2) Iteration Number I_t: Performance consistently improves with increased iterations, validating our policy optimization. On Porto, extending from 0.5k to 5k yields substantial gains: T3S-DTW improves from 0.411 to 0.454 (+10.5%), TMN-DTW from 0.582 to 0.619 (+6.4%). On Chengdu, TMN-Hausdorff improves from 0.606 to 0.698 (+15.2%). Gains diminish after 2k iterations (e.g., T3S-DTW: 0.451 at 2k vs. 0.454 at 5k), suggesting convergence around 2k iterations provides optimal cost-effectiveness.

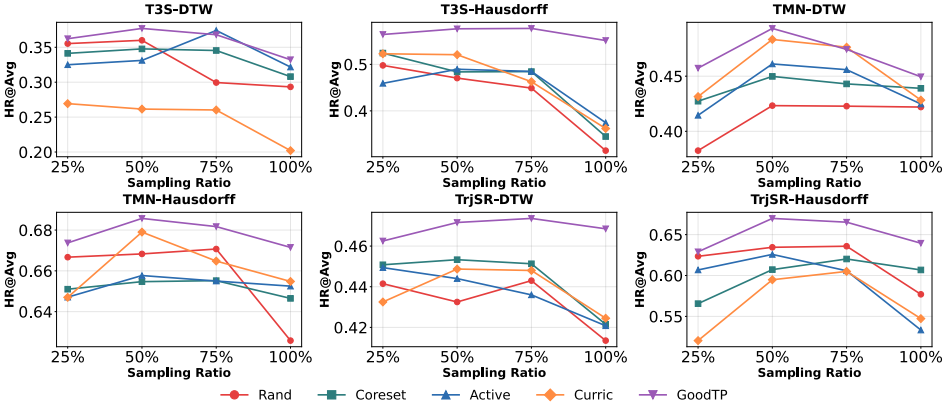
Tuning guideline. We recommend a geometric schedule for I_t (e.g., $I_t \in \{500, 1000, 2000, 5000, \ldots\}$) while monitoring the cumulative reward within the MCTS tree. The tuning should stop when the reward curve stabilizes (i.e., incremental improvements become negligible), indicating that the sampling policy has converged.

(3) Training Budget B: Budget scaling reveals critical data efficiency insights. On Porto, TMN-DTW improves from 0.513 (10k) to 0.642 (5000k) (+25.1%), while T3S-DTW shows 77.5% gains from 10k to 5000k but plateaus around 1000k. On Chengdu, TMN-Hausdorff exhibits strong scaling from 0.557 (10k) to 0.703 (5000k) (+26.2%). Importantly, substantial performance is achieved at moderate budgets (100k-1000k), with diminishing returns beyond 1000k, enabling practical deployment without exhaustive enumeration.

Tuning guideline. Our results indicate performance scales efficiently with data volume but hits diminishing returns once the budget approximates the dataset size (e.g., improvement slows after



(a) Impact of Trajectory Number



(b) Impact of Trajectory Point Sampling Ratio

Fig. 6. Evaluation of Trajectory Characteristics (Chengdu)

$B = 1000k \approx O(N)$). Therefore, we suggest starting with a budget of order $O(N)$ to ensure basic coverage, then increasing B geometrically until performance saturates.

7.6 Impact of Trajectory Characteristics

We evaluate how different selection methods perform under varying trajectory data characteristics: (1) trajectory number scaling from 25% to 100%, and (2) trajectory point sampling ratio from 25% to 100%. Due to space constraints, we report results on Chengdu (Fig. 6) and have the following observations:

(1) Performance Improves with Increased Data Volume. Performance generally improves with more trajectory data, but improvement rates vary significantly. For T3S-DTW, **GoodTP** achieves 0.227 at 25% data and 0.332 at full data (46% improvement), while random selection improves from 0.227 to 0.293 (29%). This demonstrates that intelligent selection yields greater benefits as data volume increases by prioritizing informative trajectories.

(2) Sampling Ratio Exhibits Non-Monotonic Pattern. As shown in Fig. 6b, lower trajectory point sampling ratios produce simpler trajectories that are generally easier to learn, while higher ratios preserve more details but increase learning complexity. However, extremely low ratios may discard critical trajectory information required for distinguishing different trajectories. For instance, TMN and TrjSR exhibit notably degraded performance at the 25% sampling ratio. Consequently,

most models exhibit a non-monotonic pattern: performance peaks at moderate ratios (50% to 75%) and declines at ratio 100%.

(3) Effective Selection Becomes Critical in Some Challenge Scenarios. The performance gap between **GoodTP** and baselines widens significantly in three scenarios: (i) at extremely low sampling ratios where over-simplified trajectories become difficult to distinguish, (ii) at high sampling ratios where trajectories contain more noise and complexity, and (iii) when the trajectory pool is large. This implies that simple baselines differ little from sophisticated methods only in "easy" settings with moderate complexity and small data pools.

(4) Models Exhibit Different Data Scaling Behaviors. Models show distinct sensitivity to data quantity and selection quality. TrjSR-Hausdorff improves dramatically from 0.504 (25%) to 0.640 (100%) under **GoodTP** (27% gain), while TMN-Hausdorff remains stable at 0.633-0.672 (6% gain). This suggests TrjSR requires careful data curation for optimal performance, whereas TMN's architecture extracts value from diverse patterns with less stringent selection.

(5) GoodTP Maintains Consistent Advantage Across Conditions. Despite varying data conditions, **GoodTP** consistently achieves the best or near-best performance. For sampling ratio experiments, while baselines show significant performance drops at ratio 1.0, **GoodTP** maintains robust performance with only minor degradation. This validates that our selection strategy adapts well to different trajectory characteristics without dataset-specific tuning, confirming its dataset-agnostic nature.

8 Related Work

Trajectory Similarity Learning. The evolution of trajectory similarity learning has been driven by pursuing computational efficiency and improved representation quality beyond traditional handcrafted distance metrics [35, 36]. Early deep learning approaches adopted recurrent neural networks, where t2vec [28] pioneered sequence-to-sequence architectures with grid cell-based representations for self-supervised trajectory embeddings. NEUTRAJ [48] introduced spatial attention mechanisms to capture relationships between trajectories and spatial context, while Traj2SimVec [58] enhanced efficiency by leveraging spatial indices for intelligent positive sample selection and sub-trajectory-level loss functions for fine-grained similarity modeling. To address RNN limitations in capturing complex point-level interactions, TMN [45] introduced a dual-branch architecture jointly learning trajectory embeddings and explicit point matchings, though requiring pairwise trajectory inputs. Recent advances shifted toward self-attention mechanisms, with T3S [46] combining LSTM and self-attention to model sequential dependencies and point-wise interactions, and TrajCL [8] adopting a fully attention-based encoder that adaptively learns topological and spatial features through contrastive learning. Alternative explorations include TrjSR [6], transforming trajectories into fixed-size images for CNN-based processing, and TrajGAT [49], leveraging graph neural networks over multi-level quadtree structures to model road network topology. *Despite these architectural innovations progressively improving representation capabilities, a critical limitation persists: all existing methods uniformly rely on random trajectory pair sampling to construct training data, without considering sample informativeness, difficulty, or diversity.*

Data Selection for Model Training. Effective data selection in data-centric AI encompasses two established paradigms. *Selection-before-training* approaches identify core training subsets using static metrics computed prior to model training, typically employing gradient-based computations [24, 44] or clustering-based gradient approximations [7]. For example, coreset selection methods [7] partition datasets into clusters and compute item-cluster distances to approximate gradient bounds for efficient subset selection. However, these static methods cannot adapt to evolving model capabilities and require complete upfront dataset access. In contrast, *iterative dynamic selection* methods [26, 32, 59, 61] continuously adapt selection strategies through active learning or

curriculum learning paradigms, leveraging training signals such as sample difficulty, loss gradients, or model uncertainty to progressively refine training subsets. However, these approaches suffer from myopic round-by-round optimization lacking global coherence and often require frequent model retraining and data re-evaluation.

Therefore, both paradigms assume complete dataset access and require exhaustive traversal, which is impractical for trajectory similarity learning where training pairs scale quadratically ($O(N^2)$ for N trajectories). This quadratic growth renders gradient-based coreset methods computationally infeasible. While clustering-based coreset methods could be adapted, they require redesigns for dynamic pair generation. These limitations motivate formulating data selection as a trajectory sampling problem that intelligently generates informative pairs on-the-fly without exhaustive enumeration.

Recent reinforcement learning–based data selection methods (e.g., [47]) are promising, but directly applying generic RL baselines to trajectory similarity learning faces significant computational barriers. The core challenge lies in modeling the selection policy due to the complex action space. If we formulate the action as selecting a specific trajectory or pair, the resulting discrete action space scales linearly ($O(N)$) or quadratically ($O(N^2)$) with the dataset size. Such scaling renders policy exploration and training prohibitively expensive, making competitive RL-based baselines impractical for our setting. Thus, we solve the "action space" challenge by structuring the policy as a Monte Carlo Tree Search (MCTS) on a hierarchical clustering tree with theoretical guarantees. This transforms the infeasible flat selection problem into a manageable hierarchical decision process.

9 Conclusions and Discussions

We present **GoodTP**, a principled framework addressing data selection in trajectory similarity learning. Our framework makes three contributions. First, we design a hierarchical clustering index to organize trajectories, enabling efficient cluster-based sampling. In particular, **GoodTP** adopts a modular architecture where users can substitute task-specific features for building the index without modifying the core framework. Second, we formulate selection as tree traversal policy optimization with an MCTS-based solution providing theoretical convergence guarantees. Third, we introduce cache-enhanced weighted learning through bilevel optimization, bridging the trajectory-to-pair selection gap while achieving efficiency through similarity reuse. Experiments across four models on three real-world datasets demonstrate that **GoodTP** achieves the best performance in most cases.

GoodTP is readily extensible to other domains where constructing informative training pairs is computationally expensive, such as graph similarity learning [31] and sequential recommendation systems [23]. In graph similarity learning, the data selection challenge mirrors that of trajectory representation learning. **GoodTP** addresses this by leveraging graph-level features (e.g., degree distributions, motif statistics) for hierarchical clustering, while the MCTS-based policy traverses the hierarchy to select informative pairs for training graph neural networks. For sequential recommendation, data selection is critical in contrastive learning scenarios, where choosing high-quality sequence pairs from massive user logs is essential. Modeling user sequences as item "trajectories", **GoodTP** characterizes them using behavioral features and dynamically selects informative pairs (e.g., hard negatives) to enhance training, avoiding the inefficiency of random sampling. Overall, these extensions demonstrate the broad applicability of **GoodTP** across domains with expensive or nontrivial pair construction.

Acknowledgments

This research is supported by Singapore MOE AcRF Tier-2 grant MOE-T2EP20223-0004.

References

[1] Pankaj K. Agarwal, Kyle Fox, Jiangwei Pan, and Rex Ying. 2016. Approximating Dynamic Time Warping and Edit Distance for a Pair of Point Sequences. In *SoCG*. 6:1–6:16.
[2] Stefan Atev, Grant Miller, and Nikolaos P. Papanikolopoulos. 2010. Clustering of Vehicle Trajectories. *TITS* 11, 3 (2010), 647–657.
[3] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.* 13 (2012), 281–305.
[4] Jiang Bian, Dayong Tian, Yuanyan Tang, and Dacheng Tao. 2018. A survey on trajectory clustering analysis. *arXiv* (2018).
[5] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. Comput. Intell. AI Games* 4, 1 (2012), 1–43.
[6] Hanlin Cao, Haina Tang, Yulei Wu, Fei Wang, and Yongjun Xu. 2021. On Accurate Computation of Trajectory Similarity via Single Image Super-Resolution. In *IJCNN*. 1–9.
[7] Chengliang Chai, Jiayi Wang, Nan Tang, Ye Yuan, Jiabin Liu, Yuhao Deng, and Guoren Wang. 2023. Efficient Coreset Selection with Cluster-based Methods. In *SIGKDD*. ACM, 167–178.
[8] Yanchuan Chang, Jianzhong Qi, Yuxuan Liang, and Egemen Tanin. 2023. Contrastive Trajectory Similarity Learning with Dual-Feature Attention. In *ICDE*. 2933–2945.
[9] Yanchuan Chang, Egemen Tanin, Gao Cong, Christian S Jensen, and Jianzhong Qi. 2024. Trajectory similarity measurement: An efficiency perspective. *VLDB* (2024), 2293–2306.
[10] Lei Chen and Raymond T. Ng. 2004. On The Marriage of Lp-norms and Edit Distance. In *VLDB*. 792–803.
[11] Lei Chen, M. Tamer Özsu, and Vincent Oria. 2005. Robust and Fast Similarity Search for Moving Object Trajectories. In *SIGMOD*, Fatma Özcan (Ed.). 491–502.
[12] Minxiao Chen, Haitao Yuan, Nan Jiang, Zhihan Zheng, Sai Wu, Ao Zhou, and Shangguang Wang. 2025. RLOMM: An Efficient and Robust Online Map Matching Framework with Reinforcement Learning. *Proc. ACM Manag. Data* 3, 3 (2025), 209:1–209:26.
[13] Wei Chen, Yuxuan Liang, Yuanshao Zhu, Yanchuan Chang, Kang Luo, Haomin Wen, Lei Li, Yanwei Yu, Qingsong Wen, Chao Chen, Kai Zheng, Yunjun Gao, Xiaofang Zhou, and Yu Zheng. 2024. Deep Learning for Trajectory Data Management and Mining: A Survey and Beyond. *arXiv* abs/2403.14151 (2024).
[14] Camila Leite da Silva, Lucas May Petry, and Vania Bogorny. 2019. A survey and comparison of trajectory classification methods. In *BRACIS*. 788–793.
[15] Somayeh Dodge, Robert Weibel, and Anna-Katharina Lautenschütz. 2008. Towards a taxonomy of movement patterns. *Inf. Vis.* 7, 3-4 (2008), 240–252.
[16] Thomas Eiter and Heikki Mannila. 1994. Computing discrete Fréchet distance. (1994).
[17] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*. 226–231.
[18] Ziquan Fang, Yuntao Du, Lu Chen, Yujia Hu, Yunjun Gao, and Gang Chen. 2021. E2dtc: An end to end deep trajectory clustering framework via self-training. In *ICDE*. 696–707.
[19] 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 *SIGKDD*. 347–356.
[20] Ziquan Fang, Shenghao Gong, Lu Chen, Jiachen Xu, Yunjun Gao, and Christian S Jensen. 2023. Ghost: A general framework for high-performance online similarity queries over distributed trajectory streams. *SIGMOD* (2023), 1–25.
[21] Zhenni Feng and Yanmin Zhu. 2016. A survey on trajectory data mining: Techniques and applications. *IEEE Access* 4 (2016), 2056–2067.
[22] Sylvain Gelly and David Silver. 2007. Combining online and offline knowledge in UCT. In *ICML*, Vol. 227. 273–280.
[23] Yongjing Hao, Pengpeng Zhao, Junhua Fang, Jianfeng Qu, Guanfeng Liu, Fuzhen Zhuang, Victor S. Sheng, and Xiaofang Zhou. 2024. Meta-Optimized Joint Generative and Contrastive Learning for Sequential Recommendation. In *ICDE*. IEEE, 705–718.
[24] Sungnyun Kim, Sangmin Bae, and Se-Young Yun. 2023. Coreset Sampling from Open-Set for Fine-Grained Self-Supervised Learning. In *CVPR*. 7537–7547.
[25] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.
[26] John P. Lalor and Hong Yu. 2020. Dynamic Data Selection for Curriculum Learning via Ability Estimation. In *Findings of the ACL: EMNLP 2020*. 545–555.
[27] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. 2007. Trajectory clustering: a partition-and-group framework. In *SIGMOD*. 593–604.
[28] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S. Jensen, and Wei Wei. 2018. Deep Representation Learning for Trajectory Similarity Computation. In *ICDE*. 617–628.

[29] Zhenhui Li, Jae-Gil Lee, Xiaolei Li, and Jiawei Han. 2010. Incremental Clustering for Trajectories. In *DASFAA*, Vol. 5982. 32–46.
[30] Stuart P. Lloyd. 1982. Least squares quantization in PCM. *IEEE Trans. Inf. Theory* 28, 2 (1982), 129–136.
[31] Guixiang Ma, Nesreen K. Ahmed, Theodore L. Willke, and Philip S. Yu. 2021. Deep graph similarity learning: a survey. *Data Min. Knowl. Discov.* 35, 3 (2021), 688–725.
[32] Lin Ma, Bailu Ding, Sudipto Das, and Adith Swaminathan. 2020. Active Learning for ML Enhanced Database Systems. In *SIGMOD*. ACM, 175–191.
[33] Xavier Olive, Luis Basora, Benoit Viry, and Richard Alligier. 2020. Deep trajectory clustering with autoencoders. In *ICRAT*.
[34] Maryam Shaygan, Collin Meese, Wanxin Li, Xiaoliang George Zhao, and Mark Nejad. 2022. Traffic prediction using artificial intelligence: Review of recent advances and emerging opportunities. *TR-C* 145 (2022), 103921.
[35] Jianing Si, Haitao Yuan, Nan Jiang, Minxiao Chen, Xiao Ma, and Shangguang Wang. 2025. Towards Robust Trajectory Embedding for Similarity Computation: When Triangle Inequality Violations in Distance Metrics Matter. In *ICDE*. IEEE, 1–14.
[36] Jianing Si, Haitao Yuan, Xiang Li, Nan Jiang, Xiao Ma, Guoliang Li, and Shangguang Wang. 2025. Having It Both Ways: Single Trajectory Embedding for Similarity Computation with Pairwise Learning. In *ICDE*. IEEE, 474–486.
[37] Renchu Song, Weiwei Sun, Baihua Zheng, and Yu Zheng. 2014. PRESS: A Novel Framework of Trajectory Compression in Road Networks. *VLDB* (2014), 661–672.
[38] Yueyang Su, Di Yao, Xiaolei Zhou, Yuxuan Zhang, Yunxia Fan, Lu Bai, and Jingping Bi. 2023. TripSafe: Retrieving Safety-related Abnormal Trips in Real-time with Trajectory Data. In *SIGIR*. 2446–2450.
[39] Michail Vlachos, Dimitrios Gunopulos, and George Kollios. 2002. Discovering Similar Multidimensional Trajectories. In *ICDE*. 673–684.
[40] Sheng Wang, Zhifeng Bao, J Shane Culpepper, and Gao Cong. 2021. A survey on trajectory data management, analytics, and learning. *CSUR* 54 (2021), 1–36.
[41] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Timos Sellis, and Xiaolin Qin. 2019. Fast Large-Scale Trajectory Clustering. *Proc. VLDB Endow.* 13, 1 (2019), 29–42.
[42] Senzhang Wang, Jiannong Cao, and S Yu Philip. 2020. Deep learning for spatio-temporal data mining: A survey. *TKDE* 34, 8 (2020), 3681–3700.
[43] Wei Wang, Feng Xia, Hansong Nie, Zhikui Chen, Zhiguo Gong, Xiangjie Kong, and Wei Wei. 2021. Vehicle Trajectory Clustering Based on Dynamic Representation Learning of Internet of Vehicles. *IEEE Trans. Intell. Transp. Syst.* 22, 6 (2021), 3567–3576.
[44] Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. 2024. LESS: Selecting Influential Data for Targeted Instruction Tuning. In *ICML*.
[45] Peilun Yang, Hanchen Wang, Defu Lian, Ying Zhang, Lu Qin, and Wenjie Zhang. 2022. TMN: Trajectory Matching Networks for Predicting Similarity. In *ICDE*. 1700–1713.
[46] Peilun Yang, Hanchen Wang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2021. T3S: Effective Representation Learning for Trajectory Similarity Computation. In *ICDE*. 2183–2188.
[47] Suorong Yang, Peijia Li, Furao Shen, and Jian Zhao. 2025. RL-Selector: Reinforcement Learning-Guided Data Selection via Redundancy Assessment. *CoRR* abs/2506.21037 (2025).
[48] Di Yao, Gao Cong, Chao Zhang, and Jingping Bi. 2019. Computing Trajectory Similarity in Linear Time: A Generic Seed-Guided Neural Metric Learning Approach. In *ICDE*. 1358–1369.
[49] Di Yao, Haonan Hu, Lun Du, Gao Cong, Shi Han, and Jingping Bi. 2022. TrajGAT: A Graph-based Long-term Dependency Modeling Approach for Trajectory Similarity Computation. In *KDD*. 2275–2285.
[50] Di Yao, Chao Zhang, Zhihua Zhu, Jian-Hui Huang, and Jingping Bi. 2017. Trajectory clustering via deep representation learning. In *IJCNN*. IEEE, 3880–3887.
[51] Byoung-Kee Yi, H. V. Jagadish, and Christos Faloutsos. 1998. Efficient Retrieval of Similar Time Sequences Under Time Warping. In *ICDE*. 201–208.
[52] Haitao Yuan and Gao Cong. 2026. GoodTP: Supplementary Materials and Appendix. https://github.com/yuanhaitao/GoodTP/blob/main/appendix.pdf.
[53] Haitao Yuan and Guoliang Li. 2019. Distributed In-memory Trajectory Similarity Search and Join on Road Network. In *ICDE*. 1262–1273.
[54] Haitao Yuan and Guoliang Li. 2021. A survey of traffic prediction: from spatio-temporal data to intelligent transportation. *DSE* 6, 1 (2021), 63–85.
[55] Haitao Yuan, Guoliang Li, and Zhifeng Bao. 2022. Route Travel Time Estimation on A Road Network Revisited: Heterogeneity, Proximity, Periodicity and Dynamicity. *Proc. VLDB Endow.* 16, 3 (2022), 393–405.
[56] Haitao Yuan, Guoliang Li, Zhifeng Bao, and Ling Feng. 2020. Effective Travel Time Estimation: When Historical Trajectories over Road Networks Matter. In *SIGMOD*. ACM, 2135–2149.

[57] Dengsheng Zhang and Guojun Lu. 2004. Review of shape representation and description techniques. *Pattern Recognit.* 37, 1 (2004), 1–19.
[58] Hanyuan Zhang, Xinyu Zhang, Qize Jiang, Baihua Zheng, Zhenbang Sun, Weiwei Sun, and Changhu Wang. 2020. Trajectory Similarity Learning with Auxiliary Supervision and Optimal Matching. In *IJCAI*. 3209–3215.
[59] Jiaxuan Zhang, Haitao Yuan, Jianing Si, Nan Jiang, and Shangguang Wang. 2025. Think Twice Before Imputation: Optimizing Data Imputation Order for Machine Learning. In *ICDE*. IEEE, 1362–1374.
[60] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. 2008. Understanding mobility based on GPS data. In *UbiComp*, Vol. 344. ACM, 312–321.
[61] Zhihan Zheng, Haitao Yuan, Minxiao Chen, and Shangguang Wang. 2025. RLER-TTE: An Efficient and Effective Framework for En Route Travel Time Estimation with Reinforcement Learning. *Proc. ACM Manag. Data* 3, 1 (2025), 71:1–71:26.

Received October 2025; revised January 2026; accepted February 2026