PILOT-C: Physics-Informed Low-Distortion Optimal Trajectory Compression

Kefei Wu
Fudan University
China
wukf23@m.fudan.edu.cn

Baihua Zheng
Singapore Management University
Singapore
bhzheng@smu.edu.sg

Weiwei Sun
Fudan University
China
wwsun@fudan.edu.cn

ABSTRACT

Location-aware devices continuously generate massive volumes of trajectory data, creating demand for efficient compression. Line simplification is a common solution but typically assumes 2D trajectories and ignores time synchronization and motion continuity. We propose PILOT-C, a novel trajectory compression framework that integrates frequency-domain physics modeling with error-bounded optimization. Unlike existing line simplification methods, PILOT-C supports trajectories in arbitrary dimensions, including 3D, by compressing each spatial axis independently. Evaluated on four real-world datasets, PILOT-C achieves superior performance across multiple dimensions. In terms of compression ratio, PILOT-C outperforms CISED-W, the current state-of-the-art SED-based line simplification algorithm, by an average of 19.2%. For trajectory fidelity, PILOT-C achieves an average of 32.6% reduction in error compared to CISED-W. Additionally, PILOT-C seamlessly extends to three-dimensional trajectories while maintaining the same computational complexity, achieving a 49% improvement in compression ratios over SQUISH-E, the most efficient line simplification algorithm on 3D datasets.

PVLDB Reference Format:
Kefei Wu, Baihua Zheng, and Weiwei Sun. PILOT-C: Physics-Informed Low-Distortion Optimal Trajectory Compression. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at URL_TO_YOUR_ARTIFACTS.

1 INTRODUCTION

With the rapid proliferation of location-aware devices (e.g., autonomous vehicles, drones, and IoT sensors), massive volumes of trajectory data are being generated daily. These data are typically collected at specific sampling rates and are transmitted to servers for use in a variety of applications. However, the sheer size of raw trajectory data, often riddled with redundant high-frequency noise and physically implausible motion patterns, presents significant challenges for both storage and real-time processing [10, 17, 25, 27, 28].

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

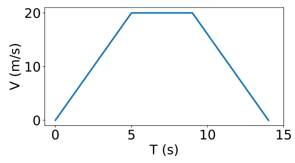
Early *lossless* trajectory compression methods, such as delta encoding [9, 27], achieved only modest compression ratios due to the inherent irregularity of trajectory data at the storage level. While suitable for archival purposes, they fall short in meeting the performance requirements of real-time systems. Consequently, attention has shifted toward *lossy* compression techniques, which strategically trade off some accuracy for higher compression efficiency.

Lossy trajectory compression fall into two main categories: *semantic-driven* and *geometric-driven* methods. Semantic-driven methods [12, 13, 30, 32, 37, 38] leverage external domain knowledge, such as road networks, to enhance compression ratios. However, their performance is heavily dependent on the availability and accuracy of such contextual information, limiting their generalizability and robustness across domains.

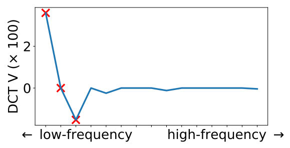
In contrast, geometric-driven methods are knowledge-agnostic, simpler to implement, and typically provide error-bounded approximations [7, 8, 10, 19–21, 24, 25]. Among them, line simplification algorithms [18] are the most widely studied. These techniques retain only the key points in a trajectory, approximating the path using simplified polylines. Representative techniques, such as CISED [17], achieve strong compression performance under a specified error bound, typically measured using the Synchronized Euclidean Distance (SED). Despite their popularity, line simplification approaches exhibit several fundamental limitations: 1) Trajectories are typically treated as geometric curves, abstracted from time. As a result, temporal dynamics (e.g., velocity changes, acceleration) are either ignored or represented only at retained points. 2) High-frequency noise (e.g., GPS jitter) is often retained, while meaningful low-frequency motion patterns (e.g., smooth acceleration or uniform motion) are not effectively preserved. 3) Although constrained by maximum error, their average error is often high, resulting in a distorted representation of the overall trajectory. 4) Most existing methods do not optimize storage structures, leading to inefficiencies in real-world deployment despite good algorithmic performance.

To overcome these challenges, we propose a novel frequency-domain approach to trajectory compression that fundamentally differs from traditional line simplification. Rather than approximating a trajectory with fewer points, we model the trajectory as a multi-dimensional time-series signal and transform it into the frequency domain using the Discrete Cosine Transform (DCT), a technique widely used in image, audio, and video compression due to its ability to separate meaningful low-frequency signals from noise-dominated high-frequency components [4, 15, 34].

This transformation reveals a key insight: trajectory data, especially from physical systems, exhibit strong temporal and spatial continuity. Objects tend to move smoothly, with gradual changes in velocity and direction, rather than abrupt or oscillatory movements.



(a) Velocity-Time curve



(b) DCT result

Figure 1: The velocity-time curve of an object moving in common motion, and the DCT result of the velocity array.

As such, the high-frequency components of the DCT-transformed trajectory often represent noise or minor irregularities. These components can be selectively discarded with minimal impact on the perceived shape or dynamics of the motion, enabling high compression ratios with low distortion. Figure 1a shows a common velocity-time graph in which an object accelerates from rest, maintains a constant speed, and then decelerates to a stop. After applying DCT, as shown in Figure 1b, the velocity signal is decomposed into a few dominant low-frequency components labeled by cross and many minor high-frequency ones.

To operationalize this insight, we propose PILOT-C (*Physics-Informed Low-Distortion Optimal Trajectory Compression*), a new framework that integrates physics modeling of motion with frequency-domain signal processing and error-bounded optimization. Our contributions are as follows:

- **Frequency-domain, physics-informed compression.** We are the first to exploit the physical structure of motion for trajectory compression. By applying DCT to each spatial dimension independently, we capture and retain the dominant low-frequency components that encode key motion behaviors. This enables us to compress trajectories effectively by removing high-frequency components that correspond to noise or physically implausible transitions.
- **Scalability to higher dimensions.** Unlike many existing algorithms—such as CISED—that are inherently restricted to 2D spatial data, PILOT-C operates independently on each dimension. This design enables natural extension to higher-dimensional trajectories (e.g., 3D), such as those from aerial drones or motion-capture systems, without redesigning the core algorithm.
- **Optimized storage and practical integration.** We optimize the storage structure used for storing the compressed representation, making PILOT-C both theoretically sound and practically deployable. This storage structure is general and can be integrated into existing line simplification algorithms to enhance their compression ratios in real-world applications.
- **Comprehensive experimental evaluation.** We evaluate PILOT-C on four real-world datasets (nuPlan, GeoLife, Mopsi, GeoLife-3D) against leading methods (CISED-S, CISED-W, SQUISH). Under the same SED constraint, it improves compression rate by 19.2% and reduces average SED error by 32.6%, preserving trajectory structure more effectively.

The rest of this paper is organized as follows. Section 2 introduces foundational concepts and background. Section 3 presents our trajectory compression framework PILOT-C. Section 4 reports the experimental results and analysis. Section 5 reviews related work, and Section 6 concludes this paper.

2 PRELIMINARIES

In this section, we introduce the core of our algorithm: the Discrete Cosine Transform (DCT). We then present a point encoding technique, which is applied in both the traditional line simplification algorithms and our proposed PILOT-C to reduce storage.

2.1 Discrete Cosine Transform (DCT)

The Discrete Cosine Transform (DCT) [1] is a real-valued, orthogonal transformation widely used in signal processing for data compression. Among its variants, the Type-II DCT (DCT-II) is the most common. For a one-dimensional discrete signal X_n, where $n = 0, 1, \ldots, N-1$, the DCT-II is defined as:

$$C_k = \alpha_k \sum_{n=0}^{N-1} X_n \cos\left(\frac{(2n+1)k\pi}{2N}\right), k = 0, 1, \ldots, N-1 \quad (1)$$

where normalization factor $\alpha_0 = \sqrt{\frac{1}{N}}$ and $\alpha_k = \sqrt{\frac{2}{N}}$ for $k > 1$.

The Inverse Discrete Cosine Transform (IDCT) reconstructs the original sequence X_n :

$$X_n = \sum_{k=0}^{N-1} \alpha_k C_k \cos\left(\frac{(2n+1)k\pi}{2N}\right), n = 0, 1, \ldots, N-1 \quad (2)$$

To better suit our application, in our implementation, 1) for DCT, we set $\alpha_k = 2$ for $k \geq 1$; and 2) for IDCT, we use $\alpha_k = \frac{1}{N}$. We leave α_0 unmodified, since C_0 is always zero in our algorithm. These modifications remove the need for square root and division operations in IDCT.

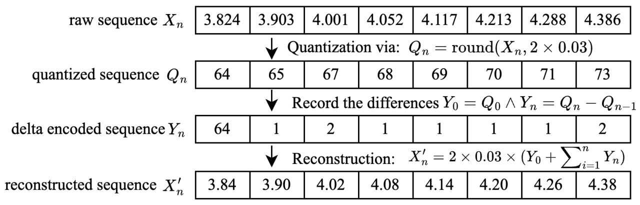
Though DCT-II is theoretically lossless when paired with its inverse and computed with full precision, practical applications involve quantization - especially of high-frequency coefficients. This introduces loss and enables compression. A core advantage of the DCT is its energy compaction: it concentrates most of a signal's energy into a few low-frequency coefficients, making it well-suited for compressing highly correlated data by retaining significant components and discarding negligible ones.

To reduce computational complexity from $O(n^2)$ to $O(n \log n)$, we use fast DCT algorithms [2, 22] that map the DCT to a Fast Fourier Transform (FFT). This involves symmetrically extending an N-point signal to $2N$ points, applying FFT, and extracting the scaled real part to obtain the DCT coefficients.

2.2 Point quantization and encoding

While trajectory simplification has been well-studied, the efficient storage of simplified points remains underexplored, often leading to suboptimal compression in practice. To address this, we propose a point quantization and encoding scheme inspired by delta encoding for more effective storage of compressed signals. This general-purpose scheme can be seamlessly integrated with existing line simplification algorithms, reducing their storage overhead and enabling fair, storage-aware comparisons with our approach.

Given a floating-point signal sequence X_n (see Figure 2), we first apply quantization using a predefined error bound ϵ_d. A rounding function, round$(X_n, 2\epsilon_d)$, maps each value X_n to the nearest multiple of $2\epsilon_d$, producing an integer X'_n. The original value can be re-constructed approximately via $2\epsilon_d \times X'_n$, with a construction error bounded by ϵ_d, i.e., $|X_n - X'_n| \leq \epsilon_d$. For example, with $\epsilon_d = 0.03$, the value $x = 3.824$ becomes 64, while $64\times(2\times\epsilon_d) = 3.84$, within ±0.03 of the original. The resulting integer sequence Q_n is



raw sequence X_n | 3.824 | 3.903 | 4.001 | 4.052 | 4.117 | 4.213 | 4.288 | 4.386
Quantization via: $Q_n = \text{round}(X_n, 2 \times 0.03)$
quantized sequence Q_n | 64 | 65 | 67 | 68 | 69 | 70 | 71 | 73
Record the differences $Y_0 = Q_0 \wedge Y_n = Q_n - Q_{n-1}$
delta encoded sequence Y_n | 64 | 1 | 2 | 1 | 1 | 1 | 1 | 2
Reconstruction: $X'_n = 2 \times 0.03 \times (Y_0 + \sum_{i=1}^{n} Y_n)$
reconstructed sequence X'_n | 3.84 | 3.90 | 4.02 | 4.08 | 4.14 | 4.20 | 4.26 | 4.38

Figure 2: Example of Quantization ($\epsilon_d = 0.03$)

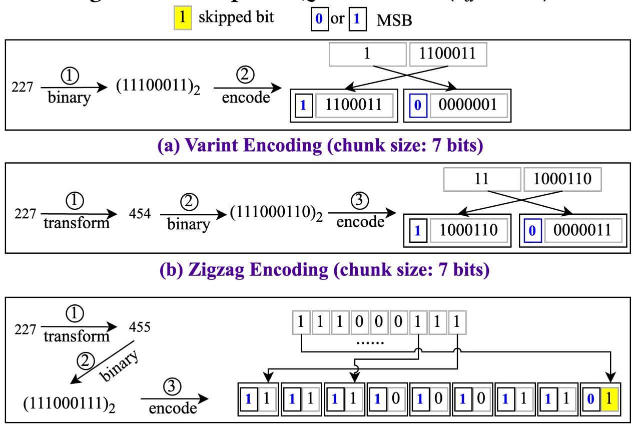


Figure 3: Examples and explanations of Varint, Zigzag, and our Enhanced Zigzag for input number 227.

delta-encoded as $Y_0 = Q_0$ and $Y_n = Q_n - Q_{n-1}$ for $n \geq 1$, shown in Figure 2. If the raw sequence X_n contains only integers, quantization can be skipped, and we simply store the first value and differences from the previous values for subsequent values for a more compact representation.

To further compress Y_n, we apply Varint encoding, a variable-length binary encoding scheme optimized for integers, and its variant Zigzag encoding for signed numbers. Varint splits each integer into 7-bit chunks, with each chunk stored in a byte. The most significant bit (MSB) of each byte acts as a continuation flag: a MSB = 1 indicates that more bytes follow, and MSB = 0 marks the final byte in the sequence. For example, the integer 227 (binary 11100011) is split into two chunks: 1 and 1100011. Varint encodes these in reverse order: **1**$\underline{\text{110 0011}}$ (MSB = 1, value = 99) and **0**$\underline{\text{000 0001}}$ (MSB = 0, value = 128), as shown in Figure 3(a). Decoding sums these values to recover 227 (= 99 + 128).

To efficiently encode signed integers, we use Zigzag encoding, which remaps signed integers to unsigned ones. It transforms non-negative numbers to even numbers ($n \longrightarrow 2n$) and negative ones to odd numbers ($n \longrightarrow 2|n| - 1$). This mapping converts small-magnitude numbers - whether positive or negative - to small unsigned integers, which Varint encodes compactly. In our example shown in Figure 3(b), Zigzag transforms input number 227 to $2 \times 227 = 454$ and then encodes 454 with Varint.

While traditional Varint + Zigzag encoding works well in general, it has two key limitations. First, traditional implementations fix the chunk size (e.g., seven bits in our examples), which may not suit all datasets. We address this by introducing a tunable chunk length parameter l, configurable at the algorithm's initialization to better adapt to different datasets. Second, standard implementations introduce redundancy: since the final byte in the encoding sequence must have MSB = 0, meaning sequences ending in an all-zero byte are wasted. This redundancy becomes more prominent with small chunk size (e.g., 1 bit), where all encoded sequences will end in '01' where 0 is MSB, and 1 is the real value.

To reduce this redundancy, we propose an enhanced Zigzag encoding variant for specific use cases. This version shifts the mapped values to eliminate zeros entirely: non-negative numbers are mapped as $n \longrightarrow 2n + 1$ and negative ones as ($n \longrightarrow 2|n|$). This ensures that all remapped integers begin with a binary bit '1'. When $l = 1$, the final encoded sequence always ends in '1', which can be safely omitted during storage and restored during decoding. For example, the input 227 becomes $2 \times 227 + 1 = 455$ (binary 111000111). If we set the chunk size to 1 bit, Varint encoding produces a sequence where the final bit '1' can be skipped, as shown in Figure 3(c), reducing total storage.

Table 1: Symbols and description

Notation	Description
$\mathcal{T}$	a raw trajectory that contains $\lvert\mathcal{T}\rvert$ elements, where each element (p_i, t_i) captures the position p_i at timestamp t_i
$\mathcal{S}, \mathcal{S}_{i,j}$	a signal array, the j-th signal array in i-th spatial dimension
$\mathcal{V}$	a velocity array derived based on an input signal array
C	a frequency array produced by transforming a velocity array via DCT
$\mathcal{F}$	output frequency array via rounding and encoding C
$C', \mathcal{V}', \mathcal{S}'$	arrays recovered by reconstructing a trajectory through decompressing the output frequency array $\mathcal{F}$ generated by PILOT-C
b_s	block size to segment signal sequences
dim	total number of spatial dimensions in a trajectory
ϵ	the overall error bound of a compression algorithm
ϵ_p	maximum allowable point-wise error
ϵ_f	maximum allowable frequency error
ϵ_t	required temporal precision

3 MODEL

This section introduces the PILOT-C compression framework, analyzing its correctness and complexity. We then propose a truncation procedure to further reduce storage and a post-compression validation step to correct any misalignments between the original and compressed points that exceed the error bound. Next, we describe the proposed storage structure for storing compressed trajectories and the decompression procedure. Finally, we extend PILOT-C to support trajectories with arbitrary sampling frequencies and discuss its strengths and limitations. Table 1 summarizes the frequently used notations.

3.1 Compression framework

Our compression model PILOT-C processes raw trajectory data to generate a compact representation. Without loss of generality, we represent a trajectory $\mathcal{T}$ as $\mathcal{T} = \langle (p_0, t_0), (p_1, t_1), \cdots \rangle$, where each tuple (p_i, t_i) captures the position p_i of the moving object at timestamp t_i. The structure of each point p_i is flexible. For example, for vehicles on road networks, p_i consists of longitude x_i and latitude y_i; for drones or aerial vehicles, p_i includes longitude x_i, latitude y_i, and altitude z_i. Alternatively, positions can be represented in the spatial geodetic Cartesian coordinate system using x_i, y_i, z_i.

To simplify the discussion, we first assume uniform sampling frequency, meaning the time interval Δt between consecutive points

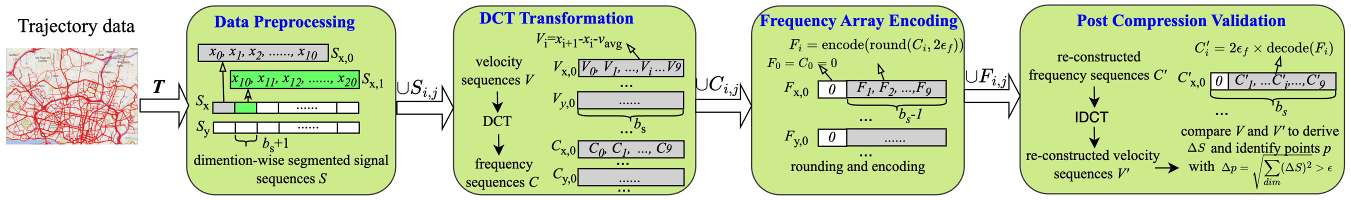

Trajectory data
T
Data Preprocessing
$x_0, x_1, x_2, \ldots\ldots, x_{10}$ $S_{x,0}$
$x_{10}, x_{11}, x_{12}, \ldots\ldots, x_{20}$ $S_{x,1}$
S_x
S_y
b_s+1
dimention-wise segmented signal sequences S
$\cup S_{i,j}$
DCT Transformation
$V_i = x_{i+1} - x_i - v_{avg}$
velocity sequences V
DCT
frequency sequences C
$V_{x,0}$ $V_0, V_1, \ldots, V_i \ldots V9$
$V_{y,0}$
b_s
$C_{x,0}$ $C_0, C_1, \ldots, C9$
$C_{y,0}$
$\cup C_{i,j}$
Frequency Array Encoding
$F_i = \text{encode}(\text{round}(C_i, 2\epsilon_f))$
$F_0 = C_0 = 0$
$F_{x,0}$ 0 $F_1, F_2, \ldots, F_9$
b_s-1
$F_{y,0}$ 0
rounding and encoding
$\cup F_{i,j}$
Post Compression Validation
$C'_i = 2\epsilon_f \times \text{decode}(F_i)$
re-constructed frequency sequences C'
$C'_{x,0}$ 0 $C'_1, \ldots C'_i \ldots, C'_9$
b_s
IDCT
re-constructed velocity sequences V'
compare V and V' to derive ΔS and identify points p with $\Delta p = \sqrt{\sum_{dim} (\Delta S)^2} > \epsilon$

Figure 4: Physics-Informed Low-Distortion Optimal Trajectory Compression framework ($b_s = 10$).

in a trajectory is constant: $\forall (p_i, t_i) \in \mathcal{T}$, $t_i - t_{i-1} = t_{i+1} - t_i = \Delta t$. We will later relax this assumption and explain how the model can accommodate trajectories with non-uniform sampling intervals.

Our model consists of three key stages: *data pre-processing*, *DCT transformation*, and *frequency array encoding*, as detailed below. Figure 4 provides a visual overview of the compression process.

Data preprocessing. We first decompose each input trajectory into signal sequences $\mathcal{S}$ along each spatial dimension. For example, given a trajectory $\mathcal{T}$, the signal sequence $\mathcal{S}$ along the x-dimension is: $\langle (x_0, t_0), (x_1, t_1), \cdots \rangle$, and similarly for y- and z-dimensions if applicable. Since time intervals are uniformly spaced, we omit them and represent the signal sequence $\mathcal{S}$ as a one-dimensional array: $\mathcal{S} = (S_0, S_1, \cdots)$, where, for example, $S_0 = x_0$, $S_1 = x_1$ and so on for the signal sequence along the x-dimension. Each signal sequence corresponding to the i-th spatial dimension is then partitioned into blocks $\mathcal{S}_{i,j}$ of based on block size b_s, which is a tunable parameter. To maintain continuity between blocks, the last point of each signal block is reused as the starting point of the next. For instance, $\mathcal{S}_{x,0} = (x_0, x_1, \cdots, x_{b_s})$, $\mathcal{S}_{x,1} = (x_{b_s}, x_{b_s+1}, \cdots, x_{2b_s})$, and so on. This segmentation improves compression efficiency, as demonstrated in our experiments. When the context is clear, we use $\mathcal{S}$ to represent a signal array of $b_s + 1$ size corresponding to a single spatial dimension.

DCT transformation. The difference between two consecutive signal values approximates the velocity of the object. Since velocity changes more smoothly than position, it is more suitable for DCT processing. Within each block and for each spatial dimension, we compute the velocity v_i as the difference between consecutive signal values: $v_i = S_{i+1} - S_i$. For example, along the x-dimension, $v_i = x_{i+1} - x_i$. We also calculate the average velocity v_{avg} within the block: $v_{avg} = \frac{\sum_{i=0}^{b_s-1} v_i}{b_s} = \frac{S_{b_s} - S_0}{b_s}$. Similar to delta compression, we construct a zero-centered velocity array $\mathcal{V} = (V_0, V_1, \cdots, V_{b_s-1})$ by subtracting the average velocity from each velocity value, where $V_0 = v_0 - v_{avg}$, $V_1 = v_1 - v_{avg}$, etc. By construction, the sum of all values in $\mathcal{V}$ is zero: $\sum V_i = 0$. Note the length of each velocity array $\mathcal{V}$ is b_s. Each velocity array $\mathcal{V}$ is then transformed using DCT, producing a frequency array $C = (C_0, C_1, \cdots, C_{b_s-1})$. Since the input $\mathcal{V}$ is zero-centered, the DC component C_0 in C that represents the average of the input signal is zero: $C_0 = 0$.

Frequency array encoding. To compress the frequency array $C = (C_0, C_1, \cdots, C_{b_s-1})$, we quantize each frequency component based on predefined frequency error threshold ϵ_f that defines the maximum allowed frequency error. Each component $C_i \in C$ is quantized as $\text{round}(C_i, 2\epsilon_f)$, where round is the rounding function that maps each value C_i to the nearest multiple of $2\epsilon_f$. Each quantized component is then encoded using the enhanced Zigzag encoding. Let $\mathcal{F}$ denote the rounded frequency array, $\mathcal{F} = (F_1, \cdots, F_{b_s-1})$ where $F_i = \text{encode}(\text{round}(C_i, 2\epsilon_f))$. Note $F_0 = C_0 = 0$ is not stored in $\mathcal{F}$ to save space.

Summary. Given a raw trajectory $\mathcal{T}$ sampled at a uniform rate, our proposed framework, PILOT-C, outputs a set of encoded frequency arrays $\cup_{i<dim, j<\lceil \frac{|\mathcal{T}|-1}{b_s} \rceil} \mathcal{F}_{i,j}$, where each $\mathcal{F}_{i,j}$ represents the encoded frequency coefficients of the j-th block along the i-th spatial dimension. Our model also stores minimal metadata necessary for accurate decompression, as detailed in Section 3.4. PILOT-C naturally supports trajectories in arbitrary dimensions by treating each spatial dimension independently in both compression and reconstruction. This makes our framework particularly suitable for modern applications involving 3D and even higher-dimensional movement data.

3.2 Correctness and complexity analyses

In the following, we prove that all reconstruction errors remain within the predefined error bound ϵ, and that the recovered points, reconstructed from the output arrays $\mathcal{F}$ of the compression model and their metadata, faithfully represent the original raw trajectory within that bound ϵ.

Consider a signal array $\mathcal{S}$ representing a trajectory along one spatial dimension within one single block, along with its corresponding velocity array $\mathcal{V}$ and frequency array C. Let $\mathcal{F}$ denote the output array produced by our compression model PILOT-C. Given $\mathcal{F}$, we reconstruct the frequency, velocity, and signal arrays, denoted as C', $\mathcal{V}'$, and $\mathcal{S}'$, respectively.

Analysis of ΔC. We first analyze the errors caused by rounding up the frequency array C as follows. Given $\mathcal{F}$ and the specific rounding function, each component C'_k in C' can be derived via Equation (3).

$$C'_k = 2\epsilon_f \text{decode}\left(F_k\right) = 2\epsilon_f \text{round}\left(C_k, 2\epsilon_f\right), k = 1, 2, \ldots, b_s - 1 \quad (3)$$

Therefore, the element-wise difference between C' and original array C is bounded by predefined allowable frequency error ϵ_f:

$$\Delta C_k = C_k - C'_k \in [-\epsilon_f, \epsilon_f), k = 1, 2, \ldots, b_s - 1 \quad (4)$$

Analysis of $\Delta \mathcal{V}$. Let $\mathcal{V}'$ denote the recovered velocity array obtained from C' via IDCT. We calculate the element-wise error in the velocity array $\mathcal{V}$, as listed in Equation (5). In this analysis, we rely on two key properties: i) $\sum V = 0$ and $C_0 = C'_0 = 0$; and ii) $\alpha'_i = \frac{1}{b_s}$, as stated in Section 2.1.

$$\begin{aligned} \Delta V_k = V_k - V'_k &= \sum_{i=0}^{b_s-1} \alpha'_i \left(C_i - C'_i\right) \cos\left(\frac{(2k+1)\, i\pi}{2b_s}\right) \\ &= \sum_{i=1}^{b_s-1} \frac{\Delta C_i \cos\left(\frac{(2k+1) i\pi}{2b_s}\right)}{b_s}, k = 0, 1, \ldots, b_s - 1 \end{aligned} \quad (5)$$

Analysis of $\Delta \mathcal{S}$. Next, we derive the element-wise difference between each recovered signal value $S'_k \in \mathcal{S}'$ and its original signal value $S_k \in \mathcal{S}$. Recall that $S_k = S_0 + \sum_{i=0}^{k-1} V_i$, $S'_k = S'_0 + \sum_{i=0}^{k-1} V'_i$. Therefore, the difference $\Delta S_k = S_k - S'_k$ can be expressed as $S_0 - S'_0 + \sum_{i=0}^{k-1} \Delta V_i$. Since $S_0 - S'_0 = 0$ (the starting value of each signal array is stored as part of the essential block-level metadata, see Section 3.4), we simplify ΔS_k via Equation (6).

$$\begin{aligned}\Delta S_k = \sum_{i=0}^{k-1} \Delta V_i &= \sum_{i=0}^{k-1} \sum_{j=1}^{b_s-1} \frac{\Delta C_j \cos\left(\frac{(2i+1)j\pi}{2b_s}\right)}{b_s} \\ &= \sum_{j=1}^{b_s-1} \frac{\Delta C_j \sum_{i=0}^{k-1} \cos\left(\frac{(2i+1)j\pi}{2b_s}\right)}{b_s} \\ &= \sum_{j=1}^{b_s-1} \frac{\Delta C_j \sin\left(\frac{kj\pi}{b_s}\right)}{2b_s \sin\left(\frac{j\pi}{2b_s}\right)}, k = 1, \dots, b_s\end{aligned} \tag{6}$$

Proof of Equation (6) In this formula, we use

$$\sum_{i=0}^{k-1} \cos\left(\frac{(2i+1)j\pi}{2b_s}\right) = \frac{\sin\left(\frac{kj\pi}{b_s}\right)}{\sin\left(\frac{j\pi}{2b_s}\right)} \tag{7}$$

This is because by Product-to-Sum formulas, we have:

$$\cos(a)\sin(b) = \frac{1}{2}\left(\sin(a+b) - \sin(a-b)\right) \tag{8}$$

Let $x = \frac{j\pi}{2b_s}$, we can obtain:

$$\begin{aligned}2\sum_{i=0}^{k-1} \cos((2i+1)x)\sin(x) &= \sum_{i=0}^{k-1}\left(\sin((2i+2)x) - \sin(2ix)\right) \\ &= \sin(2kx)\end{aligned} \tag{9}$$

We assume that each ΔC_k is independently and uniformly distributed. This implies that ΔS_k is a sum of k independent, uniformly distributed variables. From the properties of the uniform distribution, the variance of each ΔC_k is: $\mathrm{Var}\left(\Delta C_j\right) = \frac{\epsilon_f^2}{3}$. Since the expectation of ΔS_k is zero, its variance can be calculated as the sum of the variances of each uniformly distributed variable, as detailed in Equation (10).

$$\begin{aligned}\mathrm{Var}(\Delta S_k) &= \sum_{j=1}^{b_s-1} \mathrm{Var}\left(\frac{\Delta C_j \sin\left(\frac{kj\pi}{b_s}\right)}{2b_s \sin\left(\frac{j\pi}{2b_s}\right)}\right) \\ &= \sum_{j=1}^{b_s-1} \left(\frac{\sin\left(\frac{kj\pi}{b_s}\right)}{2b_s \sin\left(\frac{j\pi}{2b_s}\right)}\right)^2 \mathrm{Var}\left(\Delta C_j\right) \\ &= \sum_{j=1}^{b_s-1} \frac{\sin^2\left(\frac{kj\pi}{b_s}\right)}{4b_s^2 \sin^2\left(\frac{j\pi}{2b_s}\right)} \frac{\epsilon_f^2}{3} = \frac{\epsilon_f^2}{12b_s^2} \sum_{j=1}^{b_s-1} \frac{\sin^2\left(\frac{kj\pi}{b_s}\right)}{\sin^2 \frac{j\pi}{2b_s}} \\ &= \frac{\left(kb_s - k^2\right)\epsilon_f^2}{6b_s^2}, k = 1, \dots, b_s\end{aligned} \tag{10}$$

Proof of Equation (10) In this formula, we use

$$\sum_{j=1}^{b_s-1} \frac{\sin^2\left(\frac{kj\pi}{b_s}\right)}{\sin^2\left(\frac{j\pi}{2b_s}\right)} = 2kb_s - 2k^2 \tag{11}$$

First, we prove that $\forall x \in \mathbb{R}, m \in \mathbb{N}^*$

$$\sum_{l=0}^{m} \frac{\sin\left(\left(l + \frac{1}{2}\right)x\right)}{\sin\left(\frac{1}{2}x\right)} = \frac{\sin^2\left(\left(\frac{m+1}{2}\right)x\right)}{\sin^2\left(\frac{1}{2}x\right)} \tag{12}$$

This is because

$$\begin{aligned}\sum_{l=0}^{m} \frac{\sin\left(\left(l+\frac{1}{2}\right)x\right)}{\sin\left(\frac{1}{2}x\right)} &= \sum_{l=0}^{m} \frac{\sin\left(\left(l+\frac{1}{2}\right)x\right)\sin\left(\frac{1}{2}x\right)}{\sin^2\left(\frac{1}{2}x\right)} \\ &= \sum_{l=0}^{m} \frac{-\frac{1}{2}\left(\cos((l+1)x) - \cos(lx)\right)}{\sin^2\left(\frac{1}{2}x\right)} \\ &= \frac{-\frac{1}{2}\left(\cos((m+1)x) - 1\right)}{\sin^2\left(\frac{1}{2}x\right)} = \frac{\sin^2\left(\left(\frac{m+1}{2}\right)x\right)}{\sin^2\left(\frac{1}{2}x\right)}\end{aligned} \tag{13}$$

Let $x = \frac{j\pi}{2b_s}$, we can obtain:

$$\frac{\sin^2\left(\frac{kj\pi}{b_s}\right)}{\sin^2\left(\frac{j\pi}{2b_s}\right)} = \sum_{l=0}^{2k-1} \frac{\sin\left(\left(l+\frac{1}{2}\right)\frac{j\pi}{b_s}\right)}{\sin\left(\frac{j\pi}{2b_s}\right)} \tag{14}$$

Next, we prove that $\forall x \in \mathbb{R}, m \in \mathbb{N}^*$

$$\frac{\sin\left(\left(m+\frac{1}{2}\right)x\right)}{\sin\left(\frac{1}{2}x\right)} = 1 + 2\sum_{l=1}^{m} \cos(lx) \tag{15}$$

This is because

$$\begin{aligned}&\left(1 + 2\sum_{l=1}^{m}\cos(lx)\right)\sin\left(\frac{1}{2}x\right) \\ &= \sin\left(\frac{1}{2}x\right) + 2\sum_{l=1}^{m}\cos(lx)\sin\left(\frac{1}{2}x\right) \\ &= \sin\left(\frac{1}{2}x\right) + \sum_{l=1}^{m}\left(\sin\left(\left(l+\frac{1}{2}\right)x\right) - \sin\left(\left(l-\frac{1}{2}\right)x\right)\right) \\ &= \sin\left(m + \frac{1}{2}\right)x\end{aligned} \tag{16}$$

So that

$$\begin{aligned}\sum_{j=1}^{b_s-1} \frac{\sin^2\left(\frac{kj\pi}{b_s}\right)}{\sin^2\left(\frac{j\pi}{2b_s}\right)} &= \sum_{j=1}^{b_s-1}\sum_{l=0}^{2k-1} \frac{\sin\left(\left(l+\frac{1}{2}\right)\frac{j\pi}{b_s}\right)}{\sin\left(\frac{j\pi}{2b_s}\right)} \\ &= \sum_{j=1}^{b_s-1}\sum_{l=0}^{2k-1}\left(1 + 2\sum_{m=1}^{l}\cos\left(\frac{mj\pi}{b_s}\right)\right) \\ &= 2\sum_{j=1}^{b_s-1}\sum_{l=0}^{2k-1}\sum_{m=1}^{l}\cos\left(\frac{mj\pi}{b_s}\right) + 2kb_s - 2k\end{aligned} \tag{17}$$

Then we need to calculate $\sum_{j=1}^{b_s-1}\sum_{l=0}^{2k-1}\sum_{m=1}^{l}\cos\left(\frac{mj\pi}{b_s}\right)$:
By exchanging the sum order, we can get:

$$\sum_{j=1}^{b_s-1}\sum_{l=0}^{2k-1}\sum_{m=1}^{l}\cos\left(\frac{mj\pi}{b_s}\right) = \sum_{l=0}^{2k-1}\sum_{m=1}^{l}\sum_{j=1}^{b_s-1}\cos\left(\frac{mj\pi}{b_s}\right) \tag{18}$$

If m is odd, then

$$\cos\left(\frac{mj\pi}{b_s}\right)+\cos\left(\frac{m\left(b_s-j\right)\pi}{b_s}\right)=0 \tag{19}$$

$$2\sum_{j=1}^{b_s-1}\cos\left(\frac{mj\pi}{b_s}\right)=\sum_{j=1}^{b_s-1}\left(\cos\left(\frac{mj\pi}{b_s}\right)+\cos\left(\frac{m\left(b_s-j\right)\pi}{b_s}\right)\right)=0 \tag{20}$$

If m is even, then

$$\begin{aligned}\sum_{j=1}^{b_s-1}\cos\left(\frac{mj\pi}{b_s}\right)&=\frac{1}{2}\left(\frac{\sin\left(\left(b_s-\frac{1}{2}\right)\frac{m\pi}{b_s}\right)}{\sin\left(\frac{m\pi}{2b_s}\right)}-1\right)\\&=\frac{1}{2}\left(\frac{\sin\left(m\pi-\frac{m\pi}{2b_s}\right)}{\sin\left(\frac{m\pi}{2b_s}\right)}-1\right)=-1\end{aligned} \tag{21}$$

Since the count of odds in 1 to l is $\lfloor\frac{l+1}{2}\rfloor$ and the count of evens in 1 to l is $\lfloor\frac{l}{2}\rfloor$, so

$$\sum_{m=1}^{l}\sum_{j=1}^{b_s-1}\cos\left(\frac{mj\pi}{b_s}\right)=\left\lfloor\frac{l+1}{2}\right\rfloor\times 0+\left\lfloor\frac{l}{2}\right\rfloor\times(-1)=-\left\lfloor\frac{l}{2}\right\rfloor \tag{22}$$

$$\sum_{l=0}^{2k-1}\sum_{m=1}^{l}\sum_{j=1}^{b_s-1}\cos\left(\frac{mj\pi}{b_s}\right)=-\sum_{l=0}^{2k-1}\left\lfloor\frac{l}{2}\right\rfloor=k-k^2 \tag{23}$$

Finally,

$$\begin{aligned}\sum_{j=1}^{b_s-1}\frac{\sin^2\left(\frac{kj\pi}{b_s}\right)}{\sin^2\left(\frac{j\pi}{2b_s}\right)}&=2\sum_{j=1}^{b_s-1}\sum_{l=0}^{2k-1}\sum_{m=1}^{l}\cos\left(\frac{mj\pi}{b_s}\right)+2kb_s-2k\\&=2kb_s-2k^2\end{aligned} \tag{24}$$

Analysis of Δp. According to the Central Limit Theorem, the sum of independent uniform variables approximates a normal distribution. Therefore, we can further analyze the error of each trajectory point in Equation (25), where $|p_k-p'_k|$ denotes the spatial distance between the original point $p_k\in\mathcal{T}$ and its reconstructed counterpart $p'_k\in\mathcal{T}'$.

$$\Delta p_k=|p_k-p'_k|=\sqrt{\sum\nolimits_{i=0}^{dim-1}\Delta S_{i,k}^2} \tag{25}$$

We can then calculate this distribution's Cumulative Distribution Function (CDF) to provide an appropriate method for selecting a proper ϵ_f. Each $S_{i,k}$ is independently, normally distributed with mean of 0 and variance of $\mathrm{Var}(\Delta S_k)$, so Δp_k follows a scaled chi-square distribution with dim degrees of freedom. Let $\sigma_k=\sqrt{\mathrm{Var}(\Delta S_k)}=\sqrt{\frac{(kb_s-k^2)\epsilon_f^2}{24b_s^2}}$. Then, the expected value and CDF of Δp_k are:

$$E\left(\Delta p_k\right)=\sigma_k\sqrt{2}\frac{\Gamma\left(\frac{dim+1}{2}\right)}{\Gamma\left(\frac{dim}{2}\right)} \tag{26}$$

$$F_{\Delta p_k}(r)=\begin{cases}2\Phi\left(\frac{r}{\sigma_k}\right)-1-\sqrt{\frac{2}{\pi}}\exp\left(-\frac{r^2}{\sigma_k^2}\right)\sum_{k=0}^{m-1}\frac{1}{(2k+1)!!}\left(\frac{r^2}{\sigma_k^2}\right)^k & ,dim=2m+1\\1-\exp\left(-\frac{r^2}{2\sigma_k^2}\right)\sum_{k=0}^{m-1}\frac{1}{k!}\left(\frac{r^2}{2\sigma_k^2}\right)^k, & dim=2m\end{cases} \tag{27}$$

Consider the point with the maximum expected error, at $p_{\frac{b_s}{2}}$. For $dim=2$, its CDF becomes

$$F_{\Delta p_{\frac{b_s}{2}}}(r)=1-\exp\left(-\frac{r^2}{2\mathrm{Var}\left(\Delta S_{\frac{b_s}{2}}\right)}\right)=1-\exp\left(-\frac{12r^2}{\epsilon_f^2}\right) \tag{28}$$

The probability of exceeding a given maximum error limit ϵ is

$$P\left(\Delta p_{\frac{b_s}{2}}>\epsilon\right)=1-F_{\Delta p_{\frac{b_s}{2}}}(\epsilon)=\exp\left(-\frac{12\epsilon^2}{\epsilon_f^2}\right) \tag{29}$$

If we choose $\epsilon_f=\frac{\epsilon}{0.6}$, then this probability is 1.3%, indicating that very few points on the trajectory will exceed the maximum error. If $\epsilon_f=\frac{\epsilon}{0.8}$, the percentage could be further reduced to 0.05%.

We can also estimate the average trajectory error as:

$$\begin{aligned}E\left(\overline{\Delta T}\right)&=\sum_{k=0}^{b_s}E\left(\Delta p_k\right)=\sqrt{2}\frac{\Gamma\left(\frac{3}{2}\right)}{\Gamma(1)}\sum_{k=0}^{b_s}\sigma_k\\&=\sqrt{2}\frac{\frac{\sqrt{\pi}}{2}}{1}\sum_{k=0}^{b_s}\sqrt{\frac{\left(kb_s-k^2\right)\epsilon_f^2}{6b_s^2}}=\frac{\sqrt{\pi}\epsilon_f}{\sqrt{12}}\sum_{k=0}^{b_s}\sqrt{\frac{kb_s-k^2}{b_s^2}}\end{aligned} \tag{30}$$

Since b_s is large,

$$\sum_{k=0}^{b_s}\sqrt{\frac{kb_s-k^2}{b_s^2}}\simeq\int_0^1\sqrt{x-x^2}=\frac{\pi}{8} \tag{31}$$

Substituting it into the above equation, we can obtain:

$$E\left(\overline{\Delta T}\right)=\frac{\sqrt{\pi}\epsilon_f}{\sqrt{48}}\sum_{k=0}^{b_s}\sqrt{\frac{kb_s-k^2}{b_s^2}}\simeq\frac{\sqrt{\pi}\epsilon_f}{\sqrt{12}}\frac{\pi}{8}=\frac{\epsilon}{0.6}\frac{\pi\sqrt{\pi}}{8\sqrt{12}}\simeq 0.335\epsilon \tag{32}$$

When $dim=3$, we can obtain similarly that $E(\overline{\Delta T})\simeq 0.426\epsilon$.

Complexity analysis. For uniform sampling data, the complexity of our model is $O(n\log(b_s))$, as the trajectory is partitioned into blocks of length b_s, with DCT and IDCT applied to each block. All other operations are linear in n. In practice, since b_s is a constant for one trajectory, the overall complexity of our model is $O(n)$ where $n=|\mathcal{T}|$ is the total number of trajectory points.

3.3 Truncation and post-compression validation

Through our empirical study, we observe that the frequency arrays $\mathcal{F}_{i,j}$ produced by DCT often contain a large number of zeros. This behavior is attributed to DCT's energy compaction property, which concentrates most of the input sequence's energy (e.g., essential information) in the first few low-frequency coefficients. This effect is particularly evident in smooth or slowly varying sequences, such as the velocity sequences generated by our model. To further reduce storage requirements, we retain only a subset of the frequency components, specifically, the first k low-frequency values, since higher-frequency components often have negligible impact on trajectory shape.

Formally, given a frequency array C output by DCT, we retain only the first $k = b_s \times r_{ret}$ elements out of the total b_s coefficients, where r_{ret} is the retention rate (as defined in Equation (33)). The resulting array $\mathcal{F}$ is then given by: $\mathcal{F} = (F_1, F_2, \cdots, F_{k-1})$, where $F_i = \text{encode}(\text{round}(C_i, 2\epsilon_f))$ for $i \in [1, k-1]$. Recall $F_0 = C_0 = 0$ is not stored in $\mathcal{F}$. While this truncation strategy significantly reduces storage requirements, it introduces approximation errors and does not inherently guarantee that the reconstructed trajectory adheres to the specified error bound ϵ. To address this limitation, we introduce a post-compression validation step. This step effectively ensures compliance with the error constraint and corrects any instances where the difference between the reconstructed and original positions at certain trajectory points exceeds ϵ.

In this step, the compressed representation $\cup\mathcal{F}_{i,j}$ is decompressed to reconstruct the trajectory $\mathcal{T}'$. We then compare $\mathcal{T}'$ with the original trajectory $\mathcal{T}$ to identify any points $p_i \in \mathcal{T}$ for which the reconstruction error exceeds the specified threshold ϵ. The decompression procedure is detailed in Section 3.5. The set of such error-prone points is defined as: $\mathcal{P}_e = \{p_i \in \mathcal{T} | |p_i - p_i'| > \epsilon\}$.

For each point $p \in \mathcal{P}_e$, we compute the dimension-wise differences between the original and reconstructed signals: $\cup_{i=0}^{dim-1} \Delta S_{i,j} = S_{i,j} - S'_{i,j}$. These differences are then scaled and quantized as: $\text{round}(\Delta S_{i,j}, 2\epsilon_d)$, where $\epsilon_d = \frac{\epsilon_p}{\sqrt{dim}}$ and ϵ_p denotes the point-wise error threshold. The quantized correction values are encoded using the enhanced Zigzag encoding, and the resulting correction data are aggregated into a correction array $\mathcal{A}_c$, defined as $\mathcal{A}_c = \{\cup_{i=0}^{dim-1} \Delta S_{i,j} \wedge t_j | p_j \in \mathcal{P}_e\}$. This correction array is stored as part of the additional metadata alongside the compressed trajectories. Although this validation step increases execution time due to the additional decompression and validation step, it effectively ensures that the final output meets the desired error tolerance.

3.4 Storage structure

The final step in the compression process involves storing all previously processed data. A visualization of the data storage structure is shown in Figure 5. Note that the structure will be adjusted to support trajectory data with non-uniform sampling intervals (see Section 3.6 for details). Consequently, we present a generalized version capable of handling arbitrary input trajectories. For each input trajectory $\mathcal{T}$, our compression model outputs the compressed frequency arrays $\cup_{i<dim, j<\lceil \frac{|\mathcal{T}|-1}{b_s} \rceil} \mathcal{F}_{i,j}$, along with following metadata.

Trajectory-level metadata. The following metadata are associated with each trajectory $\mathcal{T}$: (1) Δt - the sampling interval of the trajectory, used to reconstruct timestamps; (2) ϵ - the error bound used to compress the trajectory. This is needed both to recover values from rounded integers and to compute parameters such as ϵ_f, b_s, and r_{ret}; (3) ϵ_t - the time precision to quantize timestamps. (4) $|\mathcal{T}|$ - the total number of samples (i.e., the length) of the trajectory; (5) t_0 and p_0 - the initial timestamp and starting position of the trajectory; and (6) Correction array $\mathcal{A}_c$, which stores information about error points: the total number of error points $|\mathcal{P}_e|$, and the set of timestamp-difference pairs for each dimension at every error point $\mathcal{A}_c = \{\cup_{i=0}^{dim-1} \Delta S_{i,j} \wedge t_j | p_j \in \mathcal{P}_e\}$.

Block-level metadata. The following metadata are associated with each $\mathcal{F}_{i,j}$: (1) Ending value of each block. For each segmented signal sequence $\mathcal{S}_{i,j}$, we store its ending value, which is also the starting value of the successor sequence $\mathcal{S}_{i,j+1}$. For example, in the x-dimension, the first block's ending value is S_{b_s}, the second is S_{2b_s}, and so on. Using these, the average velocity v_{avg} of each segmented signal sequence $\mathcal{S}_{i,j}$ can be derived as $\frac{S_{(j+1)b_s} - S_{jb_s}}{b_s}$. This reflects the overall trend within the block. To save space, we do not store the absolute values of $S_{i \cdot b_s}$ directly. Instead, we store the differences between consecutive block ending values (e.g., $S_{b_s} - S_0$, $S_{2 \cdot b_s} - S_{b_s}$, etc). Any ending value $S_{i \cdot b_s}$ can be reconstructed using p_0 and the cumulative sum of these differences: $\sum_{j=0}^{i-1} (S_{(j+1)b_s} - S_{jb_s})$. (2) Number of frequency components c_F. Each truncated frequency array $\mathcal{F}_{i,j}$ may still include some trailing zeros. We further reduce storage by omitting those zeros. The counter c_F associated with $\mathcal{F}_{i,j}$ records the number of meaningful components retained in $\mathcal{F}_{i,j}$.

Quantization and encoding. To further reduce the storage cost, we apply quantization and encoding techniques, as explained in Section 2.2. For floating numbers, we convert them to signed integer via quantization. Examples include: $\text{round}(\Delta S_{i,j}, 2\epsilon_d)$ where $\epsilon_d = \frac{\epsilon_p}{\sqrt{dim}}$, $\text{round}(p_0, 2\epsilon_p)$, $\text{round}(S_{(j+1) \cdot b_s} - S_{j \cdot b_s}, 2\epsilon_d)$, and $\text{round}(t_0, \epsilon_t)$. Those signed integers are encoded via Zigzag encoding. Unsigned integers (e.g., $|\mathcal{T}|$, $\cup c_F$, and $|\mathcal{P}_e|$) are encoded using Varint.

3.5 Decompression process

Next, we describe PILOT-C's decompression process. After loading all the data according to the previously defined storage structure, we reconstruct the frequency array $C'_{i,j}$. This is done by prepending $C'_0 = 0$ and padding $(b_s - c_F - 1)$ zeros to each output frequency array $\mathcal{F}_{i,j}$ to match the original block size b_s. We then reconstruct the velocity array $\mathcal{V}'_{i,j}$ by applying ICDT to $C'_{i,j}$. Using $\mathcal{V}'_{i,j}$ and the corresponding average velocity v_{avg}, which can be derived from the corresponding ending positions $S_{(j+1)b_s}$ and S_{jb_s} along the i-th dimension and p_0, we can restore the signal array $\mathcal{S}'_{i,j}$. Finally, we recover the error points using the correction array $\mathcal{A}_c$.

3.6 Extension to non-uniform sampling data

As previously mentioned, trajectory data collected from real-world applications often exhibit non-uniform sampling frequencies, requiring specific adaptations to our model for effective processing. One specific challenge is the presence of discontinuous sub-trajectories, when the temporal or spatial gap between two consecutive trajectory points is unusually large.

To address this, we segment a raw input trajectory $\mathcal{T}$ into a set of temporally continuous sub-trajectories, denoted as $\{\mathcal{T}_i\}$. For each sub-trajectory $\mathcal{T}_i$, we then construct a uniformly sampled trajectory $\mathcal{T}_i^u$ and compress $\mathcal{T}_i^u$ via our PILOT-C.

Since our model relies on the temporal continuity of trajectories and may experience degraded performance when encountering noisy start or ending points, we begin by segmenting each input trajectory $\mathcal{T}$ into trajectory fragments $\{\mathcal{T}_i\}$, ensuring that all fragments are temporally continuous. This segmentation is based on the point-wise spatial and temporal distance between consecutive trajectory points. Specifically, we use a predefined velocity threshold v_{max} (e.g., $v_{max} = 200$ m/s for road vehicles), and a time-based condition that reflects the expected sampling interval. A new segment $\mathcal{T}_{i+1}$ is initiated at point p_j if either of the following conditions holds:

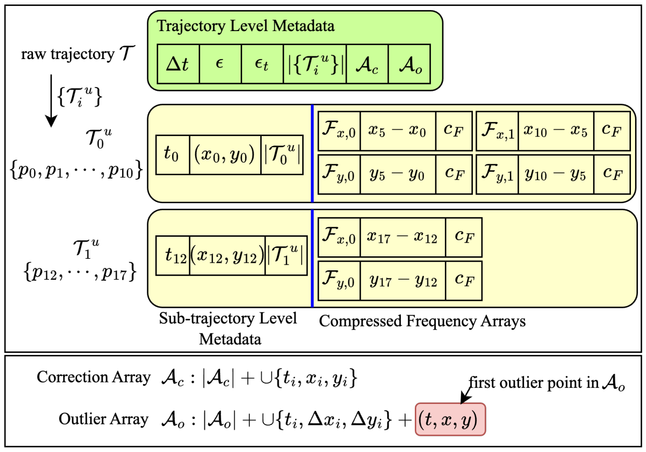


Figure 5: The adjusted storage structure of PILOT-C (b_s = 5, dim = 2).

$|p_j - p_{j-1}| > (t_j - t_{j-1}) \cdot v_{max}$ or $t_j - t_{j-1} > b_s \cdot \frac{t_j - \mathcal{T}_i.t_o}{|\mathcal{T}_i|}$, where $\mathcal{T}_i.t_o$ refers to the starting timestamp of the current trajectory fragment $\mathcal{T}_i$, and b_s is the block size that allows for tolerance to moderate irregularities. In the edge case where $|\mathcal{T}_i| = 1$, we replace $\frac{t_j - \mathcal{T}_i.t_o}{|\mathcal{T}_i|}$ with a default value Δt to handle the minimal-length fragment scenario. After segmentation, we assume, without loss of generality, that all trajectories $\{\mathcal{T}_i\}$ are temporally continuous.

We then determine the desired sampling interval Δt, which is set to the rounded average time gap between consecutive points across all temporally continuous sub-trajectories $\{\mathcal{T}_i\}$, i.e., $\Delta t = \left\lfloor \frac{\sum_i \mathcal{T}_i.t_{|\mathcal{T}_i|-1} - \mathcal{T}_i.t_0}{\sum_i |\mathcal{T}_i|} \right\rceil$. The choice of Δt introduces a trade-off in compression performance. A larger Δt results in fewer samples in the uniformly sampled trajectory $\mathcal{T}_i^u$, improving the compression ratio. However, it also increases misalignment between $\mathcal{T}_i^u$ and original $\mathcal{T}_i$, which may require storing more correction information to keep the reconstruction error within the specified error bound ϵ. In constrast, a smaller Δt better preserves fine-grained details from the original trajectory but increases the length of $\mathcal{T}_i^u$, and thus the storage cost. Although one could search for an optimal Δt by evaluating multiple values, we find that our PILOT-C delivers strong performance even with this simple average-based setting of Δt.

Next, we construct a uniformly sampled version $\mathcal{T}_i^u$ for each temporally continuous sub-trajectory $\mathcal{T}_i$, denoted as $\mathcal{T}_i^u = \langle (p_0, t_0), (p_1^u, t_0 + \Delta t), (p_2^u, t_0 + 2\Delta t), \cdots \rangle$, where (p_0, t_0) is the first point in the original trajectory $\mathcal{T}_i$. We then generate points $p_j^u \in \mathcal{T}_i^u$ at timestamp $(t_0 + j\Delta t)$ using linear interpolation. More specifically, let (p_a, t_a) and (p_{a+1}, t_{a+1}) be two consecutive points in the original trajectory $\mathcal{T}_i$ such that $(t_0 + j\Delta t) \in (t_a, t_{a+1}]$. The x-coordinate of $p_j^u \in \mathcal{T}_i^u$ is then approximated as $p_j^u.x = \frac{p_{a+1}.x - p_a.x}{t_{a+1} - t_a} \cdot (t_0 + j\Delta t - t_a) + p_a.x$. Coordinate(s) in other spatial dimension(s) are interpolated in the same manner.

Finally, we make a few key adjustments to support this process: (1) Outlier Handling: Extremely short sub-trajectories (e.g., of length 1 or 2) are treated as outliers. The corresponding points (w.r.t $\mathcal{T}$) are stored in an outlier array $\mathcal{A}_o$, sorted by their timestamps. Each point $p \in \mathcal{A}_o$ is stored as a set of signed integers representing the difference from its predecessor outlier point along each spatial dimension: $\cup_{i<dim}\text{round}(\Delta S, \frac{2\epsilon}{\sqrt{dim}})$, where ΔS indicates the difference between p and the predecessor outlier along one spatial dimension. Those signed integers are encoded via Zigzag encoding to reduce storage overhead. Similarly, timestamps are converted to unsigned integer via $\text{round}(t, \epsilon_t)$ and then encoded via Varint encoding. (2) Modified storage structure: The storage structure is adapted to support multiple sub-trajectories. Since each sub-trajectory $\mathcal{T}_i^u$ is independent, each $\mathcal{T}_i^u$ stores its own trajectory-level metadata, including start time t_0, start point p_0, and its length $|\mathcal{T}_i^u|$. Other trajectory-level metadata, including Δt, ϵ, ϵ_t, the number of sub-trajectories $|\{\mathcal{T}_i^u\}|$, outlier array $\mathcal{A}_o$, and correction array $\mathcal{A}_c$ are associated with original trajectory $\mathcal{T}$. A schematic of the updated storage structure is shown in Figure 5. (3) Timestamp encoding: Though all individual timestamps $\{t_j\}$ are treated as ground truth when decompressing the trajectory, we need to store the exact timestamps of $\mathcal{A}_o$ and $\mathcal{A}_c$. Also, for other line simplification algorithms, we also store the exact timestamps for each reserved trajectory points. Given such a sequence of timestamps, we first quantize each t_j via $\text{round}(t_j, \epsilon_t)$. We then compute the differences between successive quantized timestamps and encode these differences - positive integers - using Varint encoding. (4) Decompression adjustments: The decompression pipeline is modified accordingly. For each $\cup\mathcal{F}_{a,b}$ corresponding to a sub-trajectory $\mathcal{T}_i$, we first reconstruct a uniformly sampled sub-trajectory $(\mathcal{T}_i^u)'$ using the decompression process presented in Section 3.5. Next, we reconstruct $\mathcal{T}_i'$ by aligning $(\mathcal{T}_i^u)'$ with the original timestamps $\{t_j\}$. To be more specific, for each timestamp t_j, we locate two consecutive points $(p_b^u)'$ and $(p_{b+1}^u)'$ in $(\mathcal{T}_i^u)'$ such that $t_j \in (\mathcal{T}_i^u.t_0 + b\Delta t, \mathcal{T}_i^u.t_0 + (b+1)\Delta t]$. We then compute the interpolated position p_j' using linear interpolation between these two points. We correct error points using the correction array $\mathcal{A}_c$. Finally, we concatenate the reconstructed sub-trajectories $\mathcal{T}_i'$s and apply outlier array $\mathcal{A}_o$ to represent the complete decompressed trajectory $\mathcal{T}'$.

3.7 Strengths and limitations.

To the best of our knowledge, PILOT-C is the first trajectory compression method to exploit the physical properties of trajectories by modeling them as continuous curves. This geometric perspective enables more accurate curve fitting and effectively reduces the average error. The error introduced by PILOT-C is mathematically bounded, making it well-suited for applications that require strict accuracy guarantees. PILOT-C operates in $O(n)$ time, independent of dimensionality, ensuring strong scalability and low computational overhead even in high-dimensional data spaces. This makes PILOT-C applicable not only to conventional 2D or 3D spatial data, but also to complex, multi-dimensional trajectory datasets found in domains like biomechanics, climate modeling, and robotics. However, like other SED-based methods, PILOT-C has limitations. It relies on timestamp information, and thus cannot handle trajectories lacking temporal data. It is also not directly compatible with other distance metrics such as PED. Furthermore, PILOT-C performs best when trajectories have relatively stable sampling intervals. Its accuracy and compression effectiveness degrade when facing highly irregular sampling patterns (e.g., intervals alternating between 1 and 10 seconds), as this distorts the curve-fitting process and reduces fidelity.

Table 2: Real-life trajectory datasets

Datasets	Tra. Count	Sampling Intervals (s)	Time precision ϵ_t (s)	Δt (s)	Var(Δt)	Avg tra. length	a, b, c, d
nuPlan	1,349	0.1	0.01	0.1	1.25×10^{-8}	2,660	0.6, 20, 100, 0.04
GeoLife/GeoLife-3D	182	1-5	1	3	18.95	132,970	0.6/0.7, 0.5, 25, 1.1/0.8
Mopsi	51	2	0.001	2	5.18	149,020	0.6, 1, 25, 0.6

4 EXPERIMENTAL STUDY

In this section, we present a comprehensive experimental evaluation of our PILOT-C algorithm in comparison to state-of-the-art line simplification methods. Using four real-world trajectory datasets, we conducted a series of experiments to address the following aspects: (1) The effectiveness of various strategies proposed in our paper, including point quantization and encoding technique introduced in Section 2.2, blocking strategy in Section 3.1, and truncation strategy presented in Section 3.3. (2) The compression ratios achieved by PILOT-C compared to baselines. (3) The average error associated with each method. (4) Their performance on 3D trajectory datasets.

4.1 Experimental settings

4.1.1 Real-life trajectory datasets. To align with our assumption of uniform sampling, we first use nuPlan [5], a widely used benchmark in the autonomous driving domain. Specifically, we use one subset (i.e., **ego car trajectories**) from the test dataset, as only the ego car has a complete trajectory throughout the entire scene. This subset contains over 1,000 trajectories, each with approximately 2,660 trajectory points. It is important to note that for line simplification tasks, the number of points per trajectory plays a more critical role than the total number of trajectories in the dataset. This is because simplification quality and compression gains are primarily influenced by the density and continuity of points within individual trajectories.

To demonstrate the adaptability of our model to non-uniformly sampled datasets, we further evaluate it on GeoLife [39] and Mopsi [23]. GeoLife comprises GPS trajectory collected from 182 users with varying sampling rates, while Mopsi includes GPS trajectories from 51 users sampled at 2-second intervals. Both datasets are mainly formed by long trajectories, with each on average containing more than 130,000 trajectory points.

To evaluate the effectiveness of our model on 3D trajectory data, we enhance GeoLife by adding altitude information, resulting in a derived dataset referred to as GeoLife-3D. Although Mopsi also contains altitude information, it contains a high proportion of missing values, making it less suitable for 3D evaluation. A summary of the four datasets is provided in Table 2. Time precision is determined based on the original dataset (e.g., Mopsi stores timestamps as integer milliseconds, while GeoLife stores them as integer seconds).

4.1.2 Data preprocess. We begin by removing all the trajectory points with duplicate timestamps. Since GeoLife and Mopsi datasets provide GPS data in latitude and longitude, it is inconvenient to compute and constraint the maximum SED error directly in geographic coordinates. To address this, we select an appropriate UTM zone for each dataset and project all trajectories accordingly. The projected trajectories serve as the ground truth in our experiments.

For 3D trajectory, we convert (longitude, latitude, altitude) coordinates into Cartesian (X, Y, Z) space using the WGS84 ellipsoid model. Although GeoLife dataset has some missing altitude data, these are mostly confined to a single trajectory. Thus, we retain default values -GeoLife dataset has a default value of -777 feet for invalid altitude, and we follow this for the few missing points and construct GeoLife-3D accordingly.

4.1.3 Algorithms and implementation. We evaluate four algorithms in our experiments: PILOT-C proposed in this paper, two state-of-the-art SED-based line simplification algorithms CISED-S and CISED-W, and SQUISH, selected for its adaptability to 3D trajectories. CISED-S and CISED-W are both one-pass, error-bounded line simplification algorithms. However, CISED-S is a strong simplification algorithm, meaning all points in the compressed trajectory are drawn from the original trajectory. In contrast, CISED-W is a weak simplification algorithm, which allows the inclusion of newly generated points not present in the original trajectory. All algorithms were implemented in Python, and experiments were conducted on a 16 Core AMD Ryzen 9 5950X @ 3.40GHz.

4.1.4 Evaluation metrics. We use the following metrics to evaluate algorithm performance.

Compression ratio. Traditional trajectory simplification algorithms calculate compression rates as the ratio of retained points. Since our model uses transformed representations instead of points, we measure the compression rate based on the actual storage size. For a set of trajectories $\cup\mathcal{T}$ and their compressed version $\cup\mathcal{R}$, the compression ratio is defined as: $\sum_{\mathcal{R}_i \in \cup\mathcal{R}} \text{Size}(\mathcal{R}_i) / \sum_{\mathcal{T}_i \in \cup\mathcal{T}} \text{Size}(\mathcal{T}_i)$. Here, Size denotes the total byte size. For example, if each 2D point is stored as three double-type values (x, y, t), the storage for a trajectory $\mathcal{T}_i$ with $|\mathcal{T}_i|$ points is $192 \times |\mathcal{T}_i|$ bytes. Lower values indicate more effective compression.

Max SED error. The compressed trajectories of our data are not polygonal lines, so we cannot use the distance between line segments and points as the error metrics. Hence, error is defined as the maximum Euclidean distance between origin and reconstructed points: $\max_{\mathcal{T}_i \in \cup\mathcal{T}} \left(\max_{p_j \in \mathcal{T}_i} |p_j, p'_j| \right)$. Here, p_j and p'_j are corresponding to points in the original trajectory $\mathcal{T}_i$ and reconstructed trajectory $\mathcal{T}_i{}'$ respectively, and $|p, p'|$ is an operation to calculate the distance between two points.

This max SED error serves as the overall error bound ϵ. Due to varying sampling rates across datasets, we use different max error ranges. Specifically, we vary ϵ from 10 to 100 meters on datasets GeoLife, Mopsi, and GeoLife-3D, and vary ϵ from 1 to 10 meters on dataset nuPlan, since its sampling rate is only 0.1s and is about one-tenth of the other datasets.

Mean SED error. Defined as the average distance between corresponding original and reconstructed points:
$\left(\sum_{\mathcal{T}_i \in \cup\mathcal{T}} \sum_{p_j \in \mathcal{T}_i} |p_j, p'_j| \right) / \sum_{\mathcal{T}_i \in \cup\mathcal{T}} |\mathcal{T}_i|$.

4.1.5 Parameters. Our PILOT-C model includes three parameters: the maximum allowable frequency error ϵ_f, block size b_s, and retention rate r_{ret}. These parameters are all functions of ϵ, as defined in Equation (33). The constraint values a, b, c, and d are dataset-specific and are empirically determined through experimentation, as listed in Table 2. We evaluate the sensitivity of these parameters and their impact on performance through a series of experiments.

$$\epsilon_f = \frac{\epsilon}{a}, \quad b_s = b * \epsilon + c, \quad r_{ret} = \min\left(1, \frac{d}{\sqrt{\epsilon}}\right) \tag{33}$$

CISED requires a single parameter m, which refers to the number of sides of the regular polygon. Following the research findings reported in [17], we use $m = 16$, as increasing beyond this yields negligible gains in compression rate (<1%).

In addition to the parameters required by compression algorithms, we also introduce two additional encoding parameters, l and ϵ_p when using encoding to further reduce the storage overhead. Specifically, l refers to the length of the chunks in encoding methods Varint and Zigzag, and ϵ_p records the maximum tolerable error when rounding points to integers. The default value of l is 2 for all algorithms, while the default value of ϵ_p is 0.5ϵ for PILOT-C and 0.2ϵ for other algorithms. Note that our model utilizes ϵ_p in a way different from other algorithms. Consequently, they have different default value for ϵ_p. The impact of these parameters is also examined in our experimental analysis.

4.2 Experimental results

4.2.1 Effectiveness of point quantization/encoding, blocking, and truncation strategies. We begin by evaluating the generality and effectiveness of the quantization and encoding techniques introduced in Section 2.2, applying them to all three baseline algorithms. In Figure 6, we compare the compression ratios of the original baseline methods (labeled "No-Encode") with their enhanced versions that incorporate these techniques, tested under different ϵ values while keeping other parameters to their default settings. The results clearly demonstrate that quantization and encoding consistently improve compression ratios across all datasets and ϵ values, with an average improvement of 455%. This substantial improvement is primarily due to the high redundancy in storing compressed points using double-precision values. By reducing storage precision, the quantization and encoding methods significantly enhance compression efficiency. For the remainder of our experimental study, we only report the performance of the enhanced versions of the baseline methods, i.e., those augmented with quantization and encoding.

Next, we evaluate the impact of the enhanced Zigzag encoding technique, which is applied exclusively in PILOT-C. As shown in Figure 6, the enhanced Zigzag improves PILOT-C's compression performance by an average of 12.6%. This modest improvement stems from the relatively minor advantage of the enhanced Zigzag encoding over the original Zigzag.

PILOT-C partitions long sequences into blocks of fixed size. As discussed in Section 3.1, this segmentation improves compression efficiency. To illustrate this effect, we report the compression ratio of PILOT-C under varying block sizes b_s in Figure 9. As observed, increasing b_s from small values initially improves compression ratio. However, beyond a certain point, further increases in b_s lead to a decline in compression performance. The optimal value of b_s - at which PILOT-C achieves the highest compression ratio - varies across datasets, as it is influenced by their respective sampling rates. While empirical searches can be used to find the optimal b_s, this process must be repeated whenever the error bound ϵ changes. In contrast, we set b_s according to Equation (33). Although the parameters b and c require empirical tuning, the equation provides a consistent and effective way to estimate good values for b_s across different error bounds. As a result, PILOT-C maintains strong compression performance without re-tuning b_s for every new setting.

As discussed in Section 3.3, the frequency arrays $\mathcal{F}_{i,j}$ produced by DCT often contain many zeros, particularly in high-frequency coefficients. To improve compression, we apply truncation to retain only the first few low-frequency coefficients. To evaluate the impact of this strategy, we report the compression ratios and percentage of error points (exceed ϵ) of PILOT-C under different retention rates in Figure 7. Note that when $r_{ret} = 2^0$, no elements are removed from the frequency arrays $\mathcal{F}_{i,j}$ produced by DCT, representing the upper extreme; when $r_{ret} = 2^{-7}$, only the first 1 or 2 elements of each array are retained, resulting in the loss of important information representing the lower extreme. As expected, reducing the retention rate from 2^0 initially improves the compression ratio, while the percentage of error points remains nearly unchanged. This indicates that removing high-frequency coefficients does not significantly impact the preservation of important information. However, when the retention rate becomes too low (2^{-7} or 2^{-6}), critical information is lost from the compressed frequency arrays. This leads to a noticeable increase in error points, which in turn increases the size of the correction array $\mathcal{A}_c$, ultimately reducing overall compression efficiency. Consequently, selecting an appropriate retention rate is crucial to maintaining both compression efficiency and data fidelity. While it is possible to empirically determine an optimal r_{ret} through testing, our method, as presented in Equation(33), provides a simple and effective way to estimate a suitable r_{ret} value.

4.2.2 Compression performance. In our second set of experiments, we compare the compression performance of PILOT-C against baseline algorithms. For each experiment, we vary a single parameter while keeping all other parameters fixed at their default values.

Compression ratios vs. error bound ϵ. We first evaluate how compression performance varies with the error bound. On the GeoLife and Mopsi datasets, we vary ϵ from 10 to 100 meters, and from 1 to 10 meters on nuPlan. The results are shown in Figure 6; please focus on the solid lines. As ϵ increases, the compression ratios of all algorithms decrease across all datasets. Among the datasets, nuPlan has the best compression ratios, due to its high sampling rate. Across all tested ϵ values, PILOT-C consistently outperforms CISED-S and SQUISH-E on all datasets. It also outperforms CISED-W on nuPlan and Mopsi, and achieves comparable performance on GeoLife. Compared to nuPlan and Mopsi, GeoLife exhibits an irregular sampling pattern (i.e., $\Delta t = 3$ seconds with Var(Δt) close to 19), which affects the effectiveness of PILOT-C. On average, PILOT-C achieves compression ratios of (65.2%, 78.1%, 54.1%) and (81.8%, 100.3%, 69.6%) relative to CISED-S and CISED-W, respectively, across three datasets.

Compression ratio vs. chunk length l. To evaluate the impacts of the chunk length parameter l, which is used in the encoding phase of all algorithms, we fix the error bound $\epsilon = 50$ m for the GeoLife and Mopsi, and $\epsilon = 5$ m for nuPlan. We vary l from 1 to 7 and the results are reported in Figure 10. Across all algorithms, the compression ratio exhibits a non-monotonic trend: it initially decreases with increasing l, then increases beyond a certain point. This behavior stems from the variable effectiveness of Varint and

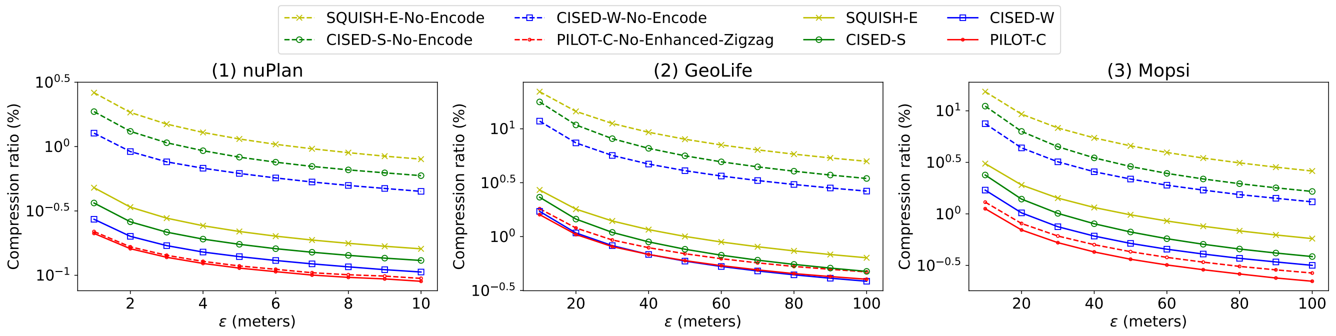


Figure 6: Compression ratios vs. ϵ.

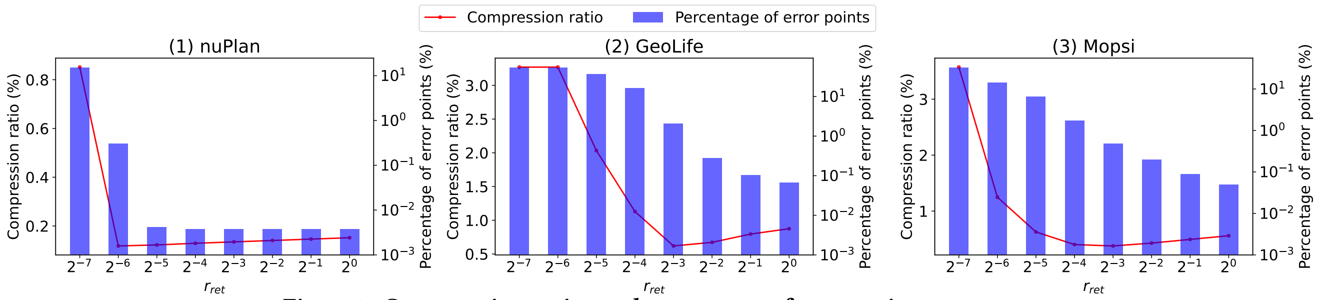


Figure 7: Compression ratios and percentage of error points vs. r_{ret}.

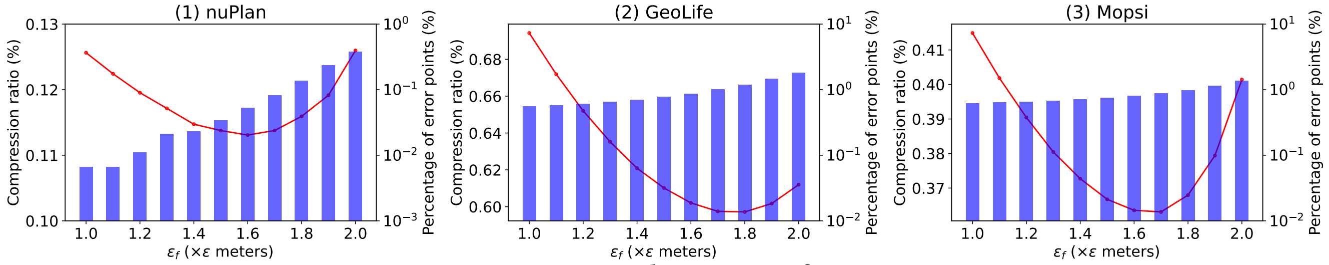


Figure 8: Compression ratios and percentage of error points vs. ϵ_f.

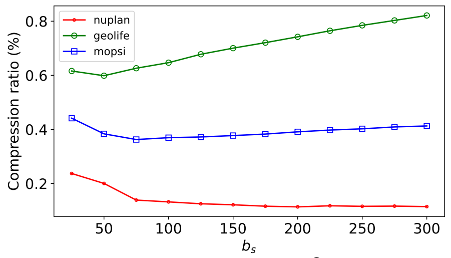


Figure 9: Compression ratios of PILOT-C vs. b_s.

Zigzag encoding, which are sensitive to chunk alignment and data distribution. The optimal l value varies across datasets, particularly in Mopsi. This variation is partly due to differences in timestamp precision. Mopsi records timestamps with 0.001-second resolution, but its average sampling interval is 2 seconds, resulting in redundancy and inefficiencies in certain encoding configurations. Given that (i) several l values yield similar compression performance, and (ii) the optimal l varies depending on ϵ, we adopt a uniform default of $l = 2$ for all algorithms in subsequent experiments to ensure consistency and simplicity.

Compression ratio vs. frequency precision ϵ_f. As analyzed in Section 3.2, the frequency precision ϵ_f directly affects the number of points on the trajectory that exceed the maximum allowable error ϵ and thus influence the size of the correction array. As ϵ_f increases, more error points emerge, as evidenced in Figure 8. On the other hand, increasing ϵ_f also reduces the value of round$(C_i, 2\epsilon_f)$, leading to shorter encoding lengths and lower storage requirements. Consequently, selecting an appropriate ϵ_f is crucial. Since the optimal values are generally consistent across all datasets, we adopt the same values for all 2D datasets, as presented in Equation(33) and Table 2, while 3D dataset has a different setting.

Compression ratio vs. point precision ϵ_p. Next, we examine the impacts of ϵ_p, which controls the precision of stored points. We fix $\epsilon = 50$ meters for GeoLife and Mopsi, and $\epsilon = 5$ meters for nuPlan. The value of ϵ_p is varied from $0.05\times\epsilon$ to $0.6\times\epsilon$. Results are shown in Figure 11. As ϵ_p increases, the compression ratios of all algorithms first decrease and then increase. This reflects the trade-off between reduced storage precision and the ability of encoding methods to efficiently compress the data. The optimal value of ϵ_p that yields the best compression is relatively consistent across datasets but varies among algorithms. This is due to how each algorithm utilizes ϵ_p. In PILOT-C, ϵ_p is only used during post-compression validation step to construct the correction array, leaving the core compression process unaffected. In contrast, CISED and SQUISH-E incorporate ϵ_p into their internal compression thresholds, directly influencing overall compression. Since multiple ϵ_p values often yield similar

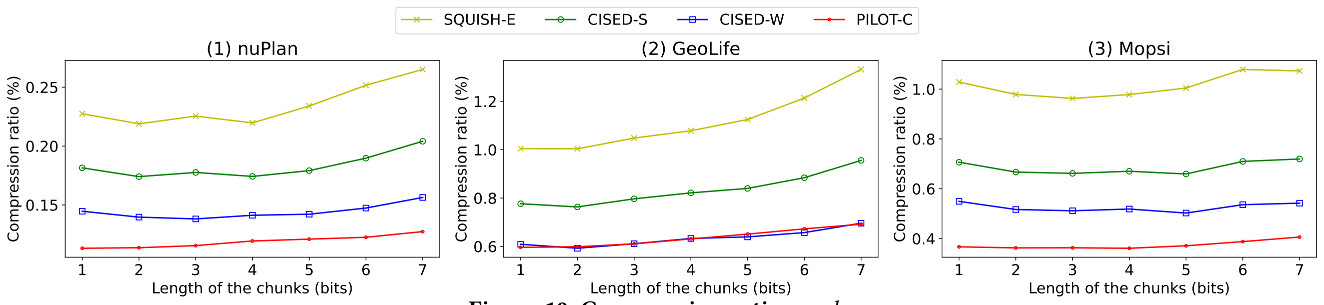


Figure 10: Compression ratios vs. l.

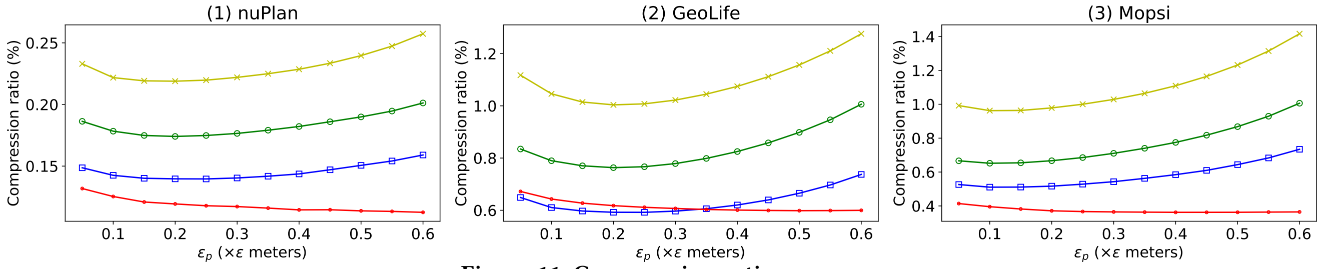


Figure 11: Compression ratios vs. ϵ_p.

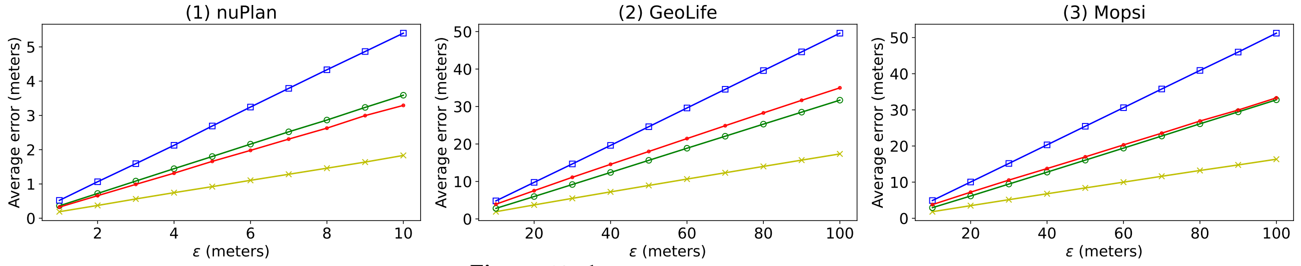
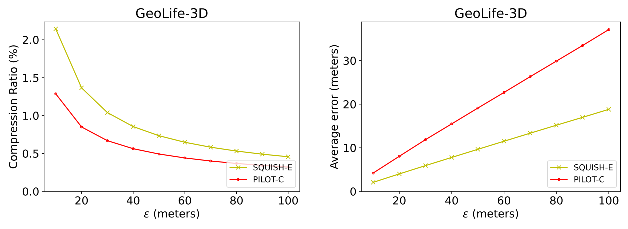


Figure 12: Average errors vs. ϵ.

compression performance, we adopt fixed defaults for simplicity in subsequent experiments: $\epsilon_p = 0.5\epsilon$ for PILOT-C and $\epsilon_p = 0.2\epsilon$ for baseline algorithms.

4.2.3 Average error comparison. In the next set of experiments, we compare the average errors of different algorithms under varying ϵ and report the results in Figure 12. As expected, the average errors increase with increasing ϵ. More specifically, the average error of each algorithm shows an approximately linear relationship with ϵ. The average error of PILOT-C is close to that of CISED-S, smaller than that of CISED-W, and larger than that of SQUISH-E. PILOT-C outperforms CISED-W because it incorporates the physical characteristics of the trajectory and treats the trajectory as a continuous curve for fitting. In addition, the experimental results are consistent with the theoretical finding presented in Equation (32), i.e., the average error of PILOT-C is approximately 0.335ϵ for 2D datasets. SQUISH-E produces the smallest errors, as it tends to retain more points, resulting in much weaker compression performance.

4.2.4 Performance on 3D trajectories. In the last set of tests, we evaluate the performance of our algorithm PILOT-C with SQUISH-E on 3D dataset. We evaluate how compression performance varies with changes of error bounds, by varying ϵ from 10 to 100 meters on GeoLife-3D. The results are reported in Figure 13. On average, PILOT-C achieves compression ratios of 67.1% relative to SQUISH-E.



(a) Compression Ratio **(b) Average Errors**

Figure 13: Performance on 3D dataset.

5 RELATED WORK

In this section, we review prior work related to trajectory compression and the usage of DCT in data compression, both of which are closely connected to this study.

5.1 Trajectory compression

Trajectory compression has been extensively studied to minimize storage overhead while ensuring bounded spatio-temporal errors. Due to the limitations of lossless compression algorithms, including low compression ratio and limited efficiency, recent research has focused on lossy compression techniques. These can be broadly categorized into two main classes: geometry-driven methods and semantics-based methods, both of which have significant limitations.

5.1.1 *Geometry-driven approaches.*

Geometry-driven approaches focus on spatial and temporal properties of trajectory data. A well-known technique is *trajectory simplification* [6, 10], also called line simplification, which selectively discards trajectory points while adhering to specific error metrics such as Perpendicular Euclidean Distance (PED), Synchronized Euclidean Distance (SED), and Direction-Aware Distance (DAD).

A foundational contribution is the "min-#" problem [18] introduced by Imai and Iri [14], which models trajectory simplification as a graph optimization problem, achieving an optimal segmentation strategy with a time complexity of $O(n^3)$. Subsequent improvements reduced this complexity to $O(n^2)$ for PED-based simplification, though scalability remains a challenge for large datasets.

To address computational constraints, various sub-optimal solutions have been proposed. Batch algorithms such as Douglas-Peucker (DP) [10] and Pavlidis's method [28]. The former splits the trajectory until it no longer needs to be cut, while the latter merges the trajectory until it no longer needs to be merged. Online algorithms do not require any prior knowledge of the entire trajectory. They process trajectory points incrementally with a local buffer. SQUISH-E [25] uses a priority queue to determine the importance of each point, removing the least significant ones while bounding the overall error. One-Pass algorithms process each trajectory point once. OPERB [19] achieves linear time complexity using PED, whereas CISED [17] introduces a spatio-temporal cone intersection technique for SED, achieving near-optimal performance.

Though effective, these trajectory simplification algorithms have common shortcomings since they can hardly balance between compression ratio, average error, efficiency, and the ability to handle multidimensional data. In contrast, our method has achieved near optimal results in all aspects by using frequency-domain, physics-informed compression, treating the trajectory as a continuous curve and optimizing the storage structure.

Other geometry-based techniques include delta compression, where each timestamp in a trajectory is encoded as a delta across time, longitude, and latitude. TrajStore [9] framework applies conventional delta compression techniques, incorporating a constant prediction model and leading-zero encoding scheme. Trajic [27] refines this approach by introducing a temporally-aware linear predictor, supporting both lossless and lossy compression. Still, these methods generally fall short in compression performance compared to trajectory simplification techniques above and also fail to solve the problem of trajectory continuity.

5.1.2 *Semantics-driven methods.*

Semantics-driven methods utilize domain-specific knowledge, such as road networks or motion patterns, to enhance trajectory compression quality. For vehicle trajectories constrained by road networks, the process typically begins with map-matching [26, 31], followed by compression. For example, COMPRESS [13, 32] identifies frequently occurring sub-trajectories and encodes them more efficiently, while UTCQ [16] focuses on the compression and queries of uncertain trajectories in road networks. Other methodologies [29] extend to custom domains by incorporating user-defined domain knowledge to gain additional compression improvement.

More recent studies have explored learning-based techniques. [11, 35, 36] RLTS [35] formulates online trajectory simplification as a sequential decision process and learns an optimal policy via reinforcement learning. S3 [11] builds a context-aware Graph Neural Network (GNN) to capture local and global mobility dependencies and proposes a novel error-distance metric called context-aware error distance (CED). While CED improves semantic fidelity, S3 shows limited gains on conventional error metrics.

Despite their effectiveness in specific domain, these methods often fail to generalize to unconstrained trajectories, such as those from pedestrians or drones, and face difficulties in balancing compression ratio with multidimensional error constraints. Comparing to these methods, our model does not require additional semantic information and can be easily transferred to various datasets.

5.2 The usage of DCT in data compression

The Discrete Cosine Transform (DCT) is a foundational technique in modern multimedia compression, renowned for its ability to concentrate signal energy into a small number of low-frequency coefficients. This property enables efficient signal representation and has led to DCT's widespread adoption in image compression (e.g., JPEG-XT [3] and JPEG [34]), audio compression (e.g., MP3 and AAC [4]), and video compression (e.g., MPEG [15] and HEVC [33]). However, these applications typically tolerate globally distributed errors, whereas trajectory compression requires strict local error bounds. This fundamental mismatch has historically limited the application of DCT to spatiotemporal data compression.

Notably absent from the literature is the use of DCT for trajectory compression. This gap stems from two inherent limitations of conventional DCT approaches: 1) Lack of local error control: The global energy compaction characteristic distributes reconstruction errors unevenly, making it difficult to guarantee maximum deviation at individual trajectory points. 2) Spatiotemporal decoupling: Standard DCT implementations process spatial and temporal dimensions independently, disrupting the intrinsic synchronization between position and timestamp in trajectory data.

Due to these limitations, trajectory compression has traditionally favored specialized geometric algorithms (e.g., DP variants) and error-bounded temporal methods (e.g., SQUISH-E), rather than frequency-domain approaches like DCT. Our work directly addresses this gap by introducing a novel synchronization-constrained DCT framework that enforces strict per-point error bounds, enabling DCT to be effectively applied to spatiotemporal trajectory data for the first time.

6 CONCLUSIONS

This paper presents PILOT-C, an error-bounded trajectory compression algorithm based on synchronous distance, which is the first model to leverage the physical significance of motion to guide trajectory compression. Experimental results demonstrate the effectiveness of PILOT-C: it achieves an average improvement of 19.2% in compression ratio compared to CISED-W, the state-of-the-art SED-based line simplification algorithm. Furthermore, PILOT-C significantly enhances fidelity, reducing the average error by 32.6% relative to CISED-W. Critically, PILOT-C extends naturally to three-dimensional trajectories while maintaining its complexity, yielding a substantial 49% gain in compression ratio over SQUISH-E, the most efficient and effective algorithm for 3D datasets.

REFERENCES

[1] Nasir Ahmed, T. Raj Natarajan, and K. R. Rao. 1974. Discrete Cosine Transform. *IEEE Trans. Computers* 23, 1 (1974), 90–93. https://doi.org/10.1109/T-C.1974.223784

[2] Yukihiro Arai, Takeshi Agui, and Masayuki Nakajima. 1988. A fast DCT-SQ scheme for images. *IEICE TRANSACTIONS (1976-1990)* 71, 11 (1988), 1095–1097.

[3] Alessandro Artusi, Rafał K Mantiuk, Thomas Richter, Philippe Hanhart, Pavel Korshunov, Massimiliano Agostinelli, Arkady Ten, and Touradj Ebrahimi. 2019. Overview and evaluation of the JPEG XT HDR image compression standard. *Journal of Real-Time Image Processing* 16 (2019), 413–428.

[4] Karlheinz Brandenburg. 1999. MP3 and AAC explained. In *Audio Engineering Society Conference: 17th International Conference: High-Quality Audio Coding*. Audio Engineering Society.

[5] Holger Caesar, Juraj Kabzan, Kok Seang Tan, Whye Kit Fong, Eric Wolff, Alex Lang, Luke Fletcher, Oscar Beijbom, and Sammy Omari. 2021. nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles. *arXiv preprint arXiv:2106.11810* (2021).

[6] Hu Cao, Ouri Wolfson, and Goce Trajcevski. 2003. Spatio-temporal data reduction with deterministic error bounds. In *Proceedings of the 2003 joint workshop on Foundations of mobile computing*. 33–42.

[7] Minjie Chen, Mantao Xu, and Pasi Franti. 2012. A fast $o(n)$ multiresolution polygonal approximation algorithm for GPS trajectory simplification. *IEEE Transactions on Image Processing* 21, 5 (2012), 2770–2785.

[8] Yukun Chen, Kai Jiang, Yu Zheng, Chunping Li, and Nenghai Yu. 2009. Trajectory simplification method for location-based social networking services. In *Proceedings of the 2009 international workshop on location based social networks*. 33–40.

[9] Philippe Cudre-Mauroux, Eugene Wu, and Samuel Madden. 2010. Trajstore: An adaptive storage system for very large trajectory data sets. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*. IEEE, 109–120.

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

[11] Ziquan Fang, Changhao He, Lu Chen, Danlei Hu, Qichen Sun, Linsen Li, and Yunjun Gao. 2023. A lightweight framework for fast trajectory simplification. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2386–2399.

[12] Ranit Gotsman and Yaron Kanza. 2015. A dilution-matching-encoding compaction of trajectories over road networks. *GeoInformatica* 19 (2015), 331–364.

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

[14] Hiroshi Imai and Masao Iri. 1986. Computational-geometric methods for polygonal approximations of a curve. *Computer Vision, Graphics, and Image Processing* 36, 1 (1986), 31–41.

[15] Didier Le Gall. 1991. MPEG: A video compression standard for multimedia applications. *Commun. ACM* 34, 4 (1991), 46–58.

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

[17] Xuelian Lin, Jiahao Jiang, Shuai Ma, Yimeng Zuo, and Chunming Hu. 2019. One-pass trajectory simplification using the synchronous Euclidean distance. *The VLDB Journal* 28, 6 (2019), 897–921.

[18] Xuelian Lin, Shuai Ma, Jiahao Jiang, Yanchen Hou, and Tianyu Wo. 2021. Error bounded line simplification algorithms for trajectory compression: An experimental evaluation. *ACM Transactions on Database Systems (TODS)* 46, 3 (2021), 1–44.

[19] Xuelian Lin, Shuai Ma, Han Zhang, Tianyu Wo, and Jinpeng Huai. 2017. One-pass error bounded trajectory simplification. *arXiv preprint arXiv:1702.05597* (2017).

[20] Jiajun Liu, Kun Zhao, Philipp Sommer, Shuo Shang, Brano Kusy, and Raja Jurdak. 2015. Bounded quadrant system: Error-bounded trajectory compression on the go. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 987–998.

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

[22] Christoph Loeffler, Adriaan Ligtenberg, and George S Moschytz. 1989. Practical fast 1-D DCT algorithms with 11 multiplications. In *International Conference on Acoustics, Speech, and Signal Processing,*. IEEE, 988–991.

[23] Radu Mariescu-Istodor and Pasi Fränti. 2017. Grid-based method for GPS route analysis for retrieval. *ACM Transactions on Spatial Algorithms and Systems (TSAS)* 3, 3 (2017), 1–28.

[24] Nirvana Meratnia and Rolf A de By. 2004. Spatiotemporal compression techniques for moving point objects. In *International Conference on Extending Database Technology*. Springer, 765–782.

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

[26] Paul Newson and John Krumm. 2009. Hidden Markov map matching through noise and sparseness. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*. 336–343.

[27] Aiden Nibali and Zhen He. 2015. Trajic: An effective compression system for trajectory data. *IEEE Transactions on Knowledge and Data Engineering* 27, 11 (2015), 3138–3151.

[28] Theodosios Pavlidis and Steven L Horowitz. 1974. Segmentation of plane curves. *IEEE transactions on Computers* 100, 8 (1974), 860–870.

[29] Kai-Florian Richter, Falko Schmid, and Patrick Laube. 2012. Semantic trajectory compression: Representing urban movement in a nutshell. *Journal of Spatial Information Science* 4 (2012), 3–30.

[30] Iulian Sandu Popa, Karine Zeitouni, Vincent Oria, and Ahmed Kharrat. 2015. Spatio-temporal compression of trajectories in road networks. *GeoInformatica* 19 (2015), 117–145.

[31] Renchu Song, Wei Lu, Weiwei Sun, Yan Huang, and Chunan Chen. 2012. Quick map matching using multi-core cpus. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*. 605–608.

[32] Renchu Song, Weiwei Sun, Baihua Zheng, and Yu Zheng. 2014. PRESS: A Novel Framework of Trajectory Compression in Road Networks. *Proc. VLDB Endow.* 7, 9 (2014), 661–672. https://doi.org/10.14778/2732939.2732940

[33] Vivienne Sze, Madhukar Budagavi, and Gary J Sullivan. 2014. High efficiency video coding (HEVC). *Integrated circuit and systems, algorithms and architectures* 39 (2014), 40.

[34] Gregory K Wallace. 1992. The JPEG still picture compression standard. *IEEE transactions on consumer electronics* 38, 1 (1992), xviii–xxxiv.

[35] Zheng Wang, Cheng Long, and Gao Cong. 2021. Trajectory simplification with reinforcement learning. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 684–695.

[36] Zheng Wang, Cheng Long, Gao Cong, and Qianru Zhang. 2021. Error-bounded online trajectory simplification with multi-agent reinforcement learning. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 1758–1768.

[37] Xiaochun Yang, Bin Wang, Kai Yang, Chengfei Liu, and Baihua Zheng. 2018. A Novel Representation and Compression for Queries on Trajectories in Road Networks. *IEEE Trans. Knowl. Data Eng.* 30, 4 (2018), 613–629. https://doi.org/10.1109/TKDE.2017.2776927

[38] Yan Zhao, Shuo Shang, Yu Wang, Bolong Zheng, Quoc Viet Hung Nguyen, and Kai Zheng. 2018. Rest: A reference-based framework for spatio-temporal trajectory compression. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2797–2806.

[39] Yu Zheng, Xing Xie, Wei-Ying Ma, et al. 2010. GeoLife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.* 33, 2 (2010), 32–39.