2026 IEEE 27th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM) | 979-8-3315-6299-1/26/$31.00 ©2026 IEEE | DOI: 10.1109/WOWMOM69805.2026.00023

# *CoST*: An Efficient and Error-Bounded Compression Algorithm for Streaming Trajectories

Zihang Xu[†], Qinqin Zhou[†], Zheng Li[‡], Ruiyuan Li[‡*], Huajun He[†*], and Yu Zheng[†§¶]
[†]*School of Computing and Artificial Intelligence*, *Southwest Jiaotong University*, Chengdu, China
[‡]*School of Big Data and Software Engineering*, *Chongqing University*, Chongqing, China
[§]*School of Cyber Engineering*, *Xidian University*, Xi'an, China
[¶]*Beijing Key Laboratory of Traffic Data Mining and Embodied Intelligence*, Beijing, China
Email: {xuzihang829, msyuzheng}@outlook.com,
{kkmi-qqz, hehuajun}@my.swjtu.edu.cn, {zhengli, ruiyuan.li}@cqu.edu.cn

***Abstract*—The widespread use of positioning devices has led to the generation of massive streaming trajectories. The transmission efficiency of such data is severely constrained by network bandwidth. To mitigate transmission pressure, trajectory compression algorithms significantly reduce data size by sacrificing negligible precision. However, existing methods either rely on batched processing with unacceptable latency or treat spatial coordinates independently, ignoring intrinsic spatio-temporal correlations. In this paper, we propose CoST, which is the first Compression of Streaming Trajectories algorithm that simultaneously satisfies strict one-pass processing, principled error bounds, and spatio-temporal awareness. Specifically, we design a novel Dual-Mode Prediction framework that adaptively switches between a lightweight linear predictor (LP) mode for stable motion and a high-precision multi-predictor (MP) mode for dynamic maneuvers. To optimize mode selection, we propose a parallel cost evaluation strategy that calculates potential bit-costs in the background, enabling purely data-driven mode switching with negligible overhead. Extensive experiments on three real-world datasets demonstrate that CoST outperforms state-of-the-art algorithms. Notably, on the WorldTrace dataset, CoST achieves a 32.6% and 60% improvement in compression ratio over the advanced streaming algorithms Serf-Qt and Serf-Xor, respectively, while maintaining an ultra-low latency of approximately 14.65 µs, validating its suitability for edge deployment.**

***Index Terms*—Trajectory Compression, Streaming Algorithms, Spatio-Temporal Data, Edge Computing, Dual-Mode Prediction**

## I. Introduction

In recent years, the rapid advancement of smart city infrastructure has generated massive **trajectory data streams** from mobile sensors, including ride-hailing vehicles, logistics fleets, and autonomous cars. These stream data have become a critical carrier of spatio-temporal information [1], empowering a variety of applications, including traffic optimization and crowd flow prediction [2]–[4], real-time fleet management and ridesharing [5], environmental monitoring [6], and itinerary recommendation [7].

This work was supported by the National Natural Science Foundation of China (Grant No. 62572086). Corresponding authors: Ruiyuan Li and Huajun He.

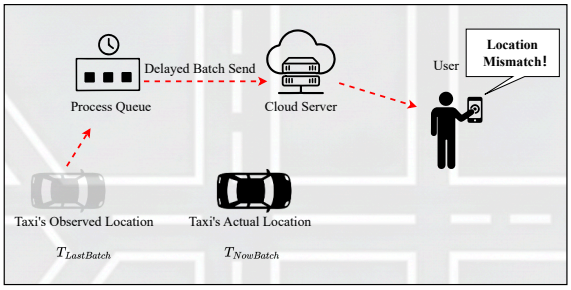


Fig. 1. Motivation: The latency problem in ride-hailing scenarios. Traditional batch or micro-batch processing leads to a "Location Mismatch," where the cloud-observed position lags behind the real-time vehicle location, causing "teleporting" effects on the map. Strict streaming is essential to ensure immediate processing and synchronization.

### A. Motivation

In time-sensitive applications like ride-hailing, the accuracy of real-time locations is paramount for dispatching. As illustrated in Fig. 1, vehicles continuously upload their coordinates to match orders. However, existing data transmission strategies often struggle to balance efficiency and latency:

- **Batch/Micro-batch Processing:** Many systems accumulate data over a window (e.g., 30 seconds) before compression and transmission. This causes a significant delay ("Location Mismatch"), where the cloud server perceives the vehicle as being at a past location. On the user's map, this manifests itself as the vehicle "teleporting" discontinuously, severely degrading the user experience and dispatching accuracy.
- **Immediate Processing Requirement:** To eliminate this lag, the system requires an immediate processing algorithm that handles data point-by-point as it arrives.

Consequently, an ideal edge-based compression algorithm must adhere to a **Strict Streaming** paradigm, defined as: *processing and transmitting each point immediately upon its arrival*, strictly prohibiting buffering or look-ahead.

### B. Limitations of Existing Approaches

The limitations of existing methods can be viewed through the lens of two dominant compression paradigms:

2770-0542/26/$31.00 ©2026 IEEE
DOI 10.1109/WoWMoM69805.2026.00023

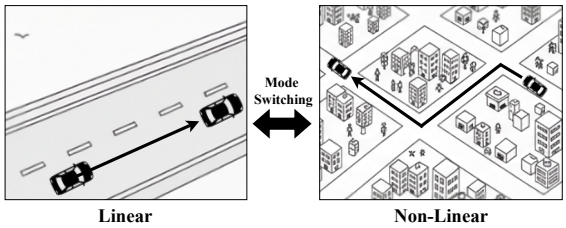


Fig. 2. Trajectory Characteristics: Mode Switching. Real-world trajectories alternate between "stable mode" (linear) and "dynamic mode" (non-linear). Relying on a single, static predictor is insufficient for optimal compression.

1. **Traditional Trajectory Compression:** Algorithms such as Douglas-Peucker (DP) [8] and SQUISH [9] are designed specifically for mobility data. However, they predominantly rely on batch processing or maintain a look-ahead buffer to optimize simplification. This inherent buffering brings unacceptable latency for real-time interactions.

2. **Generic Streaming Floating-Point Compression:** Conversely, streaming floating-point compressors like Gorilla [10] and the recent Serf family [11] achieve ultra-low latency by processing data streams element-wise. However, they treat longitude, latitude, and timestamp as independent one-dimensional streams, failing to exploit the intrinsic **spatio-temporal correlations** (e.g., kinematics constraints, such as velocity and acceleration). This results in suboptimal compression ratios compared to trajectory-aware methods.

### C. Research Challenges

To bridge the gap between these two paradigms by combining the high compression ratio of trajectory methods with the low latency of streaming methods, we must address three fundamental challenges:

- **How to effectively leverage historical spatio-temporal information under a strict streaming constraint?** Unlike generic floating-point compressors, trajectory compression must exploit the relationship between spatial displacement and time interval without look-ahead buffers. Techniques for reducing uncertainty [12] or for filling missing values [13] typically rely on future information, which is unavailable in strict streaming.
- **How to achieve low-overhead predictor selection on edge devices?** Real-world motion alternates between "stable mode" (linear) and "dynamic mode" (complex maneuvers), as shown in Fig. 2. The algorithm must adaptively switch predictors, yet edge devices lack resources for deep learning-based predictors [4]. A lightweight strategy is needed to balance compression gains against computational overhead.
- **How to efficiently encode metadata at low bit rates?** When prediction residuals are minimized, metadata (e.g., predictor flags) can dominate the bitstream. A compact encoding layout is essential to minimize this overhead and preserve the gains of accurate prediction.

### D. Proposed Solution: CoST

Therefore, we propose *CoST*, the first algorithm for the **Co**mpression of **S**treaming **T**rajectories that simultaneously satisfies strict one-pass processing, principled error bounds, and spatio-temporal awareness. Specifically, to address the challenge of non-stationary motion, *CoST* introduces a novel **Dual-Mode Prediction** framework:

- **Linear Predictor (LP) Mode**: Efficiently handles stable motion with zero metadata overhead.
- **Multi-Predictor (MP) Mode**: Employs a set of kinematic predictors to capture complex dynamics.

To select the optimal mode without stalling the pipeline, we design a **Parallel Cost Evaluation** strategy that calculates potential bit-costs in the background, enabling purely data-driven mode switching with negligible overhead.

### E. Contributions

Our main contributions are:

1) We propose *CoST*, the first error-bounded lossy compression algorithm designed for strict streaming trajectory data that simultaneously guarantees strict one-pass processing and preserves full temporal resolution. It effectively balances the high compression ratio of trajectory-specific methods with the low latency of streaming floating-point compressors.
2) We design a dual-mode predictor that adaptively switches between a lightweight LP Mode and a high-precision MP Mode, guided by a lightweight parallel cost evaluation mechanism.
3) We provide a formal definition of strict streaming error bounds and implement a data layout optimized for minimal metadata overhead.
4) Experiments on three large-scale datasets demonstrate that *CoST* outperforms state-of-the-art streaming algorithms (e.g., Serf-Qt, Serf-Xor) by 32.6%–60% in compression ratio while maintaining microsecond-level latency suitable for edge deployment.

The remainder of this paper is organized as follows: Section II reviews related work. Section III introduces the preliminaries. Section IV details the design of *CoST*. Section V presents the experimental evaluation, and Section VI concludes the paper.

## II. Related Work

Existing compression techniques can be broadly categorized into *Batch Trajectory Compression* and *Streaming Trajectory Compression* [14], [15]. This section reviews the state of the art in each category, highlighting their limitations under the strict streaming requirement.

### A. Batch Trajectory Compression

Batch methods require accumulating a complete trajectory or a large data block before processing. While they often achieve high compression ratios, their inherent latency makes them unsuitable for real-time applications.

*1) General-purpose Batch Compression:* Algorithms such as SZ [16], ZFP [17], and Machete [18] are designed for scientific floating-point data. They utilize global statistical analysis or block-based transforms to minimize data size. However, they treat trajectory coordinates as generic numerical sequences, ignoring spatial semantics, and require significant buffering, which incurs high delay.

*2) Trajectory-specific Batch Compression:* These methods exploit the geometric properties of trajectories. The classic Douglas-Peucker (DP) algorithm [8] recursively selects points based on perpendicular distance. TD-TR [19] and MRPA [20] extend this with temporal constraints. More complex frameworks like COMPRESS [21], PRESS [22], PILOT-C [23], SQUID [24], and REST [25] decompose trajectories or use reference-based encoding to maximize compression. Recent studies have also evaluated various simplification algorithms under different metrics [26], [27]. Despite their effectiveness for archival data, their dependence on full trajectory availability violates the real-time constraint.

### B. Streaming Trajectory Compression

*1) Streaming Lossless Compression (General):* Generic floating-point compressors like Gorilla [10], Chimp [28], and Elf [29] operate in a strict one-pass manner. They typically compress the XOR difference between consecutive values. While they offer ultra-low latency, their strict lossless requirement prevents them from exploiting the inherent error bounds in trajectory applications (e.g., GPS noise), resulting in a compression-ratio bottleneck.

*2) Streaming Lossy Compression (General):* The Serf family (Serf-Qt, Serf-Xor) [11] introduces error-bounded lossy compression for streaming data. They effectively smooth out noise while guaranteeing precision. However, these methods treat longitude and latitude as independent variables. By ignoring the *spatio-temporal correlations* (e.g., the physics of motion), they miss significant optimization opportunities in trajectory scenarios.

*3) Online Trajectory Simplification and Prediction:* Online simplification algorithms attempt to adapt batch methods for real-time use.

- **Window-based methods:** Algorithms like OPW [30] and SQUISH/SQUISH-E [9] maintain a sliding window or priority queue buffer. They remove the "least significant" points when the buffer is full.
- **Vector-based methods:** VOLTCom [31] uses vector processing but still relies on state vectors that may introduce micro-batching delays.

Recent frameworks like TStream [32] and online versions of direction-preserving simplification [33] have pushed the boundaries of real-time processing. However, most of these methods are actually *micro-batched* or require a *look-ahead buffer* to determine the importance of a point relative to its neighbors. This buffering still results in transmission delays (as shown in Fig. 1), failing to meet the **strict streaming** requirement of "process immediately, send immediately."

TABLE I
SUMMARY OF KEY NOTATION

| Symbol | Description |
|---|---|
| $T$ | Original **trajectory data stream** $\langle p_1, p_2, \dots \rangle$ |
| $p_i$ | The $i$-th original point $(x_i, y_i, t_i)$ |
| $\hat{p}_i$ | Predicted point derived from historical data |
| $p'_i$ | Reconstructed point available to the decompressor |
| $\epsilon$ | User-defined coordinate error bound |
| $\tau$ | Physical distance error bound (e.g., meters) |
| $\Delta t_i$ | Time interval $t_i - t_{i-1}$ |
| $r_i$ | Residual vector between predicted and actual point |
| $q_i$ | Quantized integer residual index |
| $\mathcal{L}(\cdot)$ | Bit-length function (ZigZag + Elias Gamma) |
| $\mathcal{H}(\cdot)$ | Huffman code length function |
| $\mathcal{C}$ | Accumulated **bit-cost** within a window |

While strictly streaming trajectory simplification algorithms like FBQS [34] and OP-ERB [35] offer $O(1)$ memory efficiency and error bounds, they achieve reduction by discarding samples. This loss of temporal resolution hinders downstream tasks requiring precise derivatives (e.g., instantaneous velocity analysis). In contrast, *CoST* preserves per-sample fidelity. Similarly, Dead Reckoning (DR) schemes [36] share similarities with our LP mode but often lack the fine-grained residual correction mechanism for strictly bounded per-point reconstruction.

**Summary:** *CoST* addresses these gaps by strictly adhering to the one-pass paradigm (unlike simplification methods) while explicitly modeling spatio-temporal dynamics (unlike generic floating-point compressors).

## III. PRELIMINARIES

This section defines the trajectory data model and error metrics, introduces the foundational **quantization mechanism**, and outlines the design objectives of *CoST*. Table I summarizes the key notation used throughout this paper.

### A. Definitions

*Definition 1 (Trajectory Data Stream):* We define an original **trajectory data stream** $T$ as an unbounded, temporally ordered sequence of GPS points generated by a moving object, denoted as $T = \langle p_1, p_2, \dots, p_i, \dots \rangle$. Each point $p_i$ is a tuple $(x_i, y_i, t_i)$, representing longitude, latitude, and timestamp, respectively, satisfying the monotonicity constraint $t_i < t_{i+1}$. Note that timestamps are losslessly compressed using standard delta-of-delta encoding [10]; thus, the subsequent sections focus on spatial compression.

*Definition 2 (Bounded-Error Constraint):* Lossy compression inherently introduces a deviation between the original and reconstructed data. Let $T' = \langle p'_1, p'_2, \dots \rangle$ be the reconstructed trajectory. Given a user-defined spatial error bound $\epsilon$, the compression is considered valid if and only if the following conditions are met for every $p_i$:

$$\begin{aligned} &\|p_i.\text{spatial} - p'_i.\text{spatial}\|_\infty \le \epsilon, \\ &p_i.t = p'_i.t \quad \text{(Lossless Timestamps)}. \end{aligned} \tag{1}$$

Eq. (1) ensures that the spatial error is bounded within a $2\epsilon \times 2\epsilon$ box, while timestamps are preserved losslessly to maintain temporal integrity.

**Physical to Angular Distance Conversion:**

In IoT applications, error tolerance is typically specified as a physical distance $\tau$ (e.g., 1 m). We map it to an angular bound via $\mathcal{K} \approx 1.11 \times 10^5$ m/deg. Because meters per degree vary with latitude, exact physical deviation may differ; nevertheless, the algorithm strictly enforces $\epsilon$, and latitude-aware conversion can be applied when finer geodesic fidelity is required.

### B. Foundational Components

*1) The Serf-Qt Quantization Core:* We adopt the quantization mechanism from **Serf-Qt** [11] as the arithmetic core of *CoST*. This provides a robust closed-loop framework for quantization and reconstruction.

For any scalar value $v_i$ (representing $x_i$ or $y_i$) and its prediction $\hat{v}_i$:

- **Quantization:** The prediction residual is converted into an integer index $q_i$ using a step size of $2\epsilon$:

$$q_i = \text{round}\left(\frac{v_i - \hat{v}_i}{2\epsilon}\right). \tag{2}$$

- **Reconstruction:** The decompressed value $v'_i$ is recovered on-the-fly:

$$v'_i = \hat{v}_i + q_i \times 2\epsilon. \tag{3}$$

*Proposition 1 (Error Guarantee):* As proven in [11], this mechanism strictly guarantees that $|v_i - v'_i| \leq \epsilon$ for any prediction $\hat{v}_i$. This decoupling of prediction accuracy from error control forms the mathematical foundation of our system.

**State Synchronization:** To prevent error drift, the compressor updates its history based on the reconstructed point $p'_i$ rather than the original point $p_i$. This ensures that the decompressor, which only accesses $p'_i$, remains perfectly synchronized with the encoder state.

*2) Bit-Length Estimation Function:* To support the **cost-driven** decisions detailed in Sec. IV, we define a bit-length function $\mathcal{L}(q)$. Since the quantized index $q$ is a signed integer, we use **ZigZag encoding** [37] to map it to a non-negative integer, followed by **Elias Gamma encoding** [38] for compact representation. The length function is given by:

$$\mathcal{L}(q) = \text{len}(\text{EliasGamma}(\text{ZigZag}(q))). \tag{4}$$

This function allows us to precisely quantify the compression overhead of any prediction residual.

### C. Problem Formulation

To overcome the limitations of existing methodologies, we define the problem of **streaming trajectory compression** under four stringent requirements:

1) **Strict Streaming Paradigm (One-Pass):** The algorithm must process points $p_i$ sequentially with $\mathcal{O}(1)$ time and space complexity. Decisions for $p_i$ must depend *only* on historical data $\{p_1, \ldots, p_{i-1}\}$, strictly forbidding **look-ahead** buffers.
2) **Principled Lossy Guarantee:** The system must strictly satisfy the error bound (Eq. (1)) for every data point. Heuristic filtering is not permitted unless the reconstruction error is mathematically bounded.
3) **Spatio-Temporal Joint Modeling:** Prediction models must explicitly leverage timestamps $t_i$ and intervals $\Delta t_i$ to model the spatio-temporal correlations of the trajectory (e.g., velocity and acceleration), rather than treating spatial coordinates as independent time series.
4) **Cost-Driven Adaptivity:** The algorithm must automatically detect motion pattern changes (e.g., from linear driving to complex maneuvers) and dynamically switch encoding strategies to minimize the total bit-cost $\sum \mathcal{L}(q_i)$.

## IV. The *CoST* Framework

Building upon the definitions and the Serf-Qt foundation established in Sec. III, this section introduces *CoST*, a novel algorithm designed to systematically meet the four requirements of strict streaming, principled error control, spatio-temporal joint modeling, and cost-driven adaptivity.

### A. Architecture Overview

Fig. 3 illustrates the architecture of *CoST*. To strictly adhere to the one-pass constraint (Requirement 1), the workflow is structured as a pipeline of three tightly coupled stages:

1) **Dual-Mode Prediction Module:** For each incoming point $p_i$, the system generates a prediction. Unlike static models, this module dynamically operates in one of two modes: a lightweight Linear Predictor (active in **Linear Predictor (LP) Mode**) or a complex Multi-Predictor ensemble (active in **Multi-Predictor (MP) Mode**).
2) **Cost-Driven Decision Engine:** The **Mode Controller** acts as the decision-making core. It employs a parallel cost-evaluation mechanism to simultaneously track the theoretical bit cost of both modes. Assessing the cumulative costs over a sliding window determines the most efficient mode for the subsequent data block.
3) **Quantization and Bitstream Generation:** The residual vector $r_i = p_i - \hat{p}_i$ is passed to the underlying Serf-Qt module to generate an integer quantized index $q_i$ via Eq. (2). Finally, a bitstream assembler packs the mode flags and residuals into a hierarchical data layout.

Crucially, the architecture integrates a closed-loop feedback path. Both the compressor and decompressor maintain an identical queue of reconstructed points $p'_i$ (calculated via Eq. (3)), ensuring strict state synchronization without transmitting heavy model parameters.

### B. Spatio-Temporal Predictors

To satisfy Requirement 3 (spatio-temporal joint modeling), we move beyond treating coordinates as independent variables. As shown in Fig. 4, we design predictors that explicitly

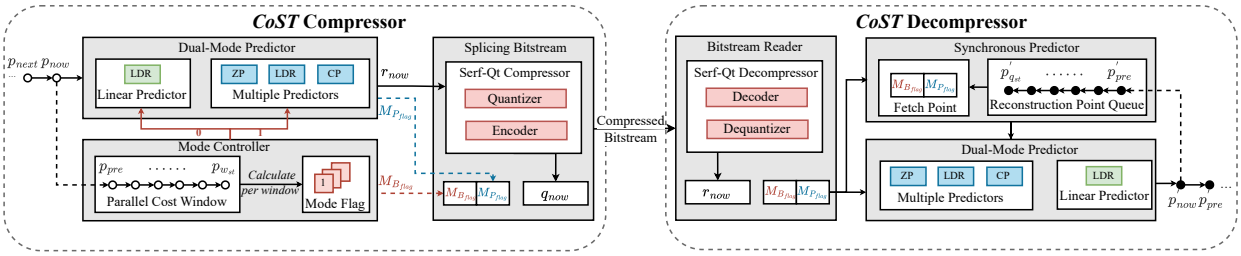


Fig. 3. The *CoST* Framework. The compressor employs a dual-mode predictor to evaluate compression costs in parallel, with a **Mode Controller** determining the optimal encoding strategy. The decompressor maintains a synchronized state to reconstruct the trajectory without explicit state-sync overhead.

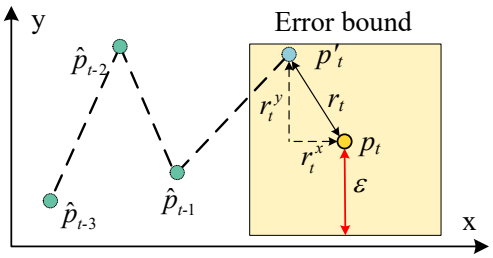


Fig. 4. Spatio-temporal prediction with error bound constraints. The predicted point $\hat{p}_i$ is derived from historical reconstructed points $p'_{i-2}$ and $p'_{i-1}$, considering the time interval $\Delta t$. The actual point $p_i$ is successfully encoded if it falls within the $\epsilon$ bounding box.

leverage the time interval $\Delta t_i = t_i - t_{i-1}$ to model motion dynamics.

Let $p'_{i-1}$ and $p'_{i-2}$ be the latest reconstructed points available in the system's state. We define the reconstructed velocity vector $v'_{i-1}$ as:

$$v'_{i-1} = \frac{p'_{i-1} - p'_{i-2}}{t_{i-1} - t_{i-2}}. \tag{5}$$

We design three operators activated within the Multi-Predictor ensemble. Note that all inputs are drawn strictly from the set of reconstructed points $\{p'\}$ to prevent error drift.

*1) Zero Predictor (ZP):* Assumes the object is stationary. This predictor is highly effective at traffic lights or during severe congestion.

$$\hat{p}_i^{(\mathrm{ZP})} = p'_{i-1}. \tag{6}$$

*2) Linear Predictor (LP):* Assumes the object maintains a constant velocity vector. This serves as the backbone for the "stable mode" (e.g., highway driving).

$$\hat{p}_i^{(\mathrm{LP})} = p'_{i-1} + v'_{i-1} \cdot \Delta t_i. \tag{7}$$

*3) Curve Predictor (CP):* Assumes constant acceleration. We estimate acceleration $a'_{i-1} = (v'_{i-1} - v'_{i-2})/(t_{i-1} - t_{i-2})$ and project the next position using a spatio-temporal kinematic model:

$$\hat{p}_i^{(\mathrm{CP})} = p'_{i-1} + v'_{i-1} \cdot \Delta t_i + \frac{1}{2} a'_{i-1} \cdot (\Delta t_i)^2. \tag{8}$$

Regardless of the chosen predictor, the final quantized index $q_i$ is calculated using Eq. (2), ensuring the error bound $\epsilon$ is strictly met.

### C. Adaptive Dual-Mode Strategy

A key observation is that minimizing the prediction error $\|p_i - \hat{p}_i\|$ does not always minimize the total bit consumption. This is because informing the decompressor which predictor was used incurs a metadata overhead. To balance precision with this overhead, *CoST* defines two mutually exclusive compression modes:

*1) LP Mode (Silent Mode):* The system is restricted to the **Linear Predictor (LP)**. **Logic:** Since the decompressor defaults to LP, no flag bits are transmitted. **Advantage:** This maximizes efficiency for linear trajectories.

*2) MP Mode (Adaptive Mode):* For each point, the system dynamically selects the best predictor $k \in \{\mathrm{ZP}, \mathrm{LP}, \mathrm{CP}\}$ that yields the minimum combined cost. **Logic:** It captures complex maneuvers but requires a predictor flag. **Advantage:** This acts as an "adaptive mode" ensuring accuracy through spatio-temporal modeling.

**Metadata Optimization via Adaptive Huffman:** To mitigate metadata overhead in MP Mode, we employ a lightweight **Adaptive Huffman Encoding** scheme for the predictor set $S = \{\mathrm{ZP}, \mathrm{LP}, \mathrm{CP}\}$. A sliding window (e.g., $N = 96$) tracks predictor frequency to assign the shortest code (1-bit) to the most frequent one. To ensure strict synchronization, the frequency table is updated after every point; in LP Mode (no flag), both sides implicitly update it assuming 'LP' usage. Ties follow a fixed priority (ZP $>$ LP $>$ CP) to guarantee identical codebooks. With fixed $|S| = 3$, this maintenance remains $\mathcal{O}(1)$ per point.

### D. Parallel Cost Evaluation and Mode Switching

To determine the optimal mode without pipeline stalls, we propose a **parallel cost evaluation** strategy. Regardless of the currently active mode, *CoST* speculatively computes the potential bit-cost for both modes for every incoming point in the background.

Let $\mathcal{L}(\cdot)$ denote the bit-length estimation function (ZigZag + Elias Gamma). Based on the quantization formula, the cost accumulation functions for a window $\mathcal{W}$ are defined as:

$$\mathcal{C}_{\mathrm{LP}} = \sum_{j \in \mathcal{W}} \mathcal{L}\left(\mathrm{round}\left(\frac{p_j - \hat{p}_j^{(\mathrm{LP})}}{2\epsilon}\right)\right). \tag{9}$$

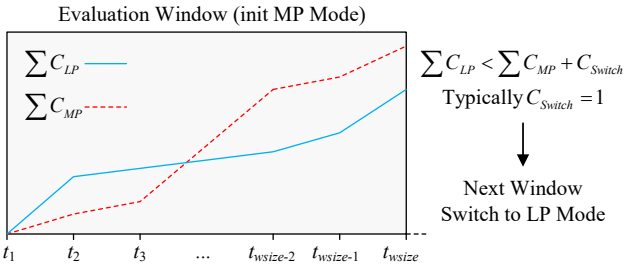


Fig. 5. The cost-driven mode-switching mechanism. The system tracks the cumulative bit cost of the **MP Mode** ($\sum C_{MP}$) and the **LP Mode** ($\sum C_{LP}$) over a sliding window. A mode switch is triggered when the potential savings exceed the switching cost.

$$\mathcal{C}_{\text{MP}} = \sum_{j \in \mathcal{W}} \left( \min_{k} \left[ \mathcal{L} \left( \text{round} \left( \frac{p_j - \hat{p}_j^{(k)}}{2\epsilon} \right) \right) + \mathcal{H}(k) \right] \right). \tag{10}$$

where $\mathcal{H}(k)$ is the dynamic Huffman code length assigned to predictor $k$ based on recent frequency.

**Overhead Justification:** Although $\mathcal{C}_{\text{MP}}$ evaluates residuals for all predictors, this relies on $O(1)$ integer arithmetic and table lookups without bitstream packing. This embodies a **compute-for-bandwidth trade-off**: marginal CPU cycles identify stable motion (enabling zero-metadata LP Mode) and significantly reduce the transmission load—the primary bottleneck at the edge.

As shown in Fig. 5, at the boundary of each evaluation window, the controller compares the cumulative costs. A mode switch is triggered for the **next window** if the potential savings exceed the switching cost $\mathcal{C}_{switch}$ (typically 1 bit):

$$mode_{new} = \begin{cases} \text{LP Mode} & \text{if } \mathcal{C}_{\text{LP}} < \mathcal{C}_{\text{MP}} + \mathcal{C}_{switch}, \\ \text{MP Mode} & \text{otherwise.} \end{cases} \tag{11}$$

### *E. Hierarchical Data Layout*

To realize the dual-mode strategy in the final output, we designed the hierarchical data layout shown in Fig. 6.

- **Stream Level:** The stream is divided into consecutive windows. A header defines global parameters, such as the error bound $\epsilon$.
- **Window Level:** Each window begins with a single bit indicating the mode: '1' for **MP Mode** and '0' for **LP Mode**.
- **Point Level:**
  - In **MP Mode**: Each point consists of an adaptive Huffman flag (1-2 bits) identifying the predictor, followed by the residual data.
  - In **LP Mode**: The flag is omitted entirely. The bitstream contains only the residuals, achieving maximum compactness.

### *F. Algorithm Description*

*1) Compression Logic:* Algorithm 1 summarizes the strict streaming process of *CoST*. The algorithm follows a macro-logic: for each arriving point, it performs parallel cost estimation, executes the best encoding strategy for the current mode, updates the shared state, and periodically re-evaluates the mode choice at window boundaries.

**Algorithm 1** *CoST* Compression Procedure

**Require:** Stream $\mathcal{P}$, error $\epsilon$, window $W$
**Ensure:** Bitstream $\mathcal{B}$
**Initialization:**
Init history $H$; $C_{MP}, C_{LP} \leftarrow 0$; $mode \leftarrow$ MP
Write $mode$ to $\mathcal{B}$ *// Header for 1st Window*
**for** each point $p_i \in \mathcal{P}$ **do**
***Phase 1: Parallel Cost Evaluation***
  Predict $\mathcal{S} = \{\hat{p}_{LP}, \hat{p}_{CP}, \hat{p}_{ZP}\}$ using $H$
  $k^* \leftarrow \arg\min_{k \in \mathcal{S}} \text{Cost}(p_i, \hat{p}_k, \text{Flag}_k)$
  $C_{MP} \leftarrow C_{MP} + \text{Cost}(p_i, \hat{p}_{k^*}, \text{Flag}_{k^*})$
  $C_{LP} \leftarrow C_{LP} + \text{Cost}(p_i, \hat{p}_{LP}, 0)$
***Phase 2: Adaptive Encoding***
  $\hat{p} \leftarrow (mode = \text{LP})\ ?\ \hat{p}_{LP} : \hat{p}_{k^*}$
  **if** $mode =$ MP **then** Write Huffman$(k^*)$ to $\mathcal{B}$
  **end if**
  $q_i \leftarrow \text{Quantize}(p_i, \hat{p}, \epsilon)$; Write $q_i$ to $\mathcal{B}$
***Phase 3: State Sync & Switching***
  $p'_i \leftarrow \text{Reconstruct}(\hat{p}, q_i)$; $H$.add$(p'_i)$
  **if** $i \equiv 0 \pmod W$ **then**
    $mode \leftarrow (C_{LP} < C_{MP} + 1)\ ?$ LP : MP
    Write $mode$ to $\mathcal{B}$ *// Header for Next Window*
    Reset $C_{LP}, C_{MP}$
  **end if**
**end for**

*2) Decompression Logic:* The decompression process is the strict inverse of compression, maintaining a prediction state identical to the compressor's. The workflow proceeds as follows: 1) **Read Mode**: At the start of a window, the 1-bit mode flag is read. 2) **Read Point**: For each point, the predictor $k$ is either identified by reading a Huffman flag (in **MP Mode**) or automatically set to LP (in **LP Mode**). 3) **Reconstruct**: The prediction $\hat{p}$ is generated using the history, and the residual $q_i$ is decoded to compute $p'_i = \hat{p} + q_i \cdot 2\epsilon$. 4) **Update**:

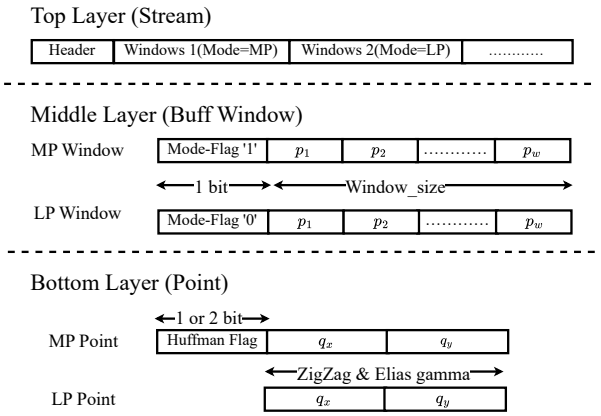


Fig. 6. Hierarchical bitstream layout of *CoST*. The structure is optimized to minimize metadata overhead. The top layer manages the stream header, the middle layer handles window-level mode flags, and the bottom layer implements adaptive encoding for individual points based on the active mode.

The reconstructed point $p'_i$ is appended to the history buffer. Because all predictions rely on $p'$, the decompressor remains perfectly synchronized without needing periodic snapshots.

**Complexity Analysis:** For each point $p_i$, *CoST* computes three constant-time predictions and cost estimates. Quantization, reconstruction, predictor selection, and adaptive-Huffman updates are all $\mathcal{O}(1)$ because the predictor set is fixed at size three. The history $\mathcal{H}$ stores only a constant number of recent points. Therefore, the time and space complexity are $\mathcal{O}(N)$ and $\mathcal{O}(1)$, respectively; larger predictor pools are left for future work.

## V. Experiments

### A. Experimental Setup

*1) Experimental Procedure:* The evaluation follows a rigorous streaming protocol. Trajectory points are read sequentially from the dataset, simulating a real-time sensor stream where future data is inaccessible. For each incoming point, we immediately measure the wall-clock time of compression and decompression. Finally, we verify that the reconstructed coordinates strictly satisfy the user-defined error bound $\epsilon$ against the ground truth.

*2) Datasets:* We utilize three diverse, large-scale trajectory datasets: **WorldTrace** [39][1] (880 million points, 2.45 million trajectories from 70 countries, 1s sampling), which serves as a stress test for model robustness under global variations; **GeoLife** [40][2] (24.88 million points from 182 users in Beijing, 1–5s sampling), capturing broad human motion patterns; and **WX-Taxi**[3] (3,673 trajectories in western China, approx. 3s sampling), representing high-density urban road network constraints.

*3) Baselines:* We benchmark *CoST* against 11 algorithms categorized as follows: **Streaming Lossless:** Chimp [28] and Elf [29], representing the state-of-the-art in generic streaming compression. **Batched Lossy:** SZ2 [16] and Machete [18], representing high-ratio batch compressors. **Streaming Lossy:** Serf-Qt and Serf-Xor [11], the current leaders in error-bounded streaming compression. **Trajectory Simplification:** The classic DP [8] algorithm (batch), and four online micro-batch algorithms: OPW-TR [30], Dead Reckoning (D.R.), SQUISH-E [9], and VOLTCom [31].

*4) Environment and Metrics:* All experiments were conducted on a workstation equipped with an Apple M4 processor and 16 GB RAM, running macOS. Implementations were in C++ to ensure fair comparison. We uniformly set the physical error bound to 1 meter (approx. $0.00001°$) for all lossy algorithms. Trajectory compression baselines used their standard error configurations, while generic floating-point compressors encoded longitude, latitude, and timestamp independently. Batch and micro-batch sizes were fixed at 50 to simulate a low-latency streaming environment. For *CoST*, the sliding window size was set to 96 (approx. 1.6 minutes), and the switching cost $\mathcal{C}_{switch}$ was fixed to 1 bit.

We evaluate performance using three key metrics:

- **Compression Ratio (CR):** The ratio of compressed size to original size (lower is better).
- **Processing Time:** The average processing time per point for compression and decompression ($\mu$s/point).
- **End-to-End Latency:** The cumulative delay from point generation to availability at the receiver, encompassing compression, transmission, and decompression.

To facilitate reproducibility, the source code of *CoST* has been made publicly available on GitHub[4].

### B. Overall Performance

*1) Compression Ratio:* Table II details the compression ratio performance. The results demonstrate that *CoST* consistently outperforms all baselines. Reported compression ratios account for spatial residuals, mode flags, and predictor flags (timestamps are excluded). On average, *CoST* achieves a compression ratio (CR) of 0.081, surpassing the leading streaming lossy algorithms Serf-Qt (0.102) and Serf-Xor (0.160). Compared to batch lossy algorithms like Machete (0.253) and SZ2 (0.579), *CoST* exhibits superior efficiency by effectively exploiting **spatio-temporal correlations**.

Notably, on the WorldTrace dataset, *CoST* achieves a CR of 0.057 (approx. 17.57×), representing a 32.6% improvement over Serf-Qt (0.084) and 60% over Serf-Xor (0.143). This result underscores the advantage of the **LP/CP** prediction strategy over treating coordinates as independent streams. Furthermore, *CoST* outperforms all trajectory simplification methods. Unlike these baselines, which compress by discarding samples, *CoST* preserves full temporal resolution through bounded per-sample reconstruction.

*2) Efficiency:* Tables III and IV report the compression and decompression processing time based on precise measurements with Dead Code Elimination (DCE) protection. Note: Processing times represent the core algorithmic latency measured via high-resolution counters, excluding system I/O, to isolate the method's computational efficiency.

As shown in Table III, Serf-Qt and DP are the fastest (approx. 0.01–0.03 $\mu$s/point) due to their simplicity. *CoST* incurs a justifiable overhead (avg. **0.120** $\mu$s/point) from parallel cost evaluation and mode switching. While slightly slower than Serf-Qt, it remains much faster than SZ2 (0.763 $\mu$s on WorldTrace, $\approx 16\times$ slower). Compared to Machete (0.052 $\mu$s), *CoST* achieves comparable speed on GeoLife and WorldTrace while delivering far superior CR. On WX-Taxi, frequent mode switching raises time to 0.235 $\mu$s, yet yields a meaningful CR gain (0.130 vs. 0.147). Its $O(1)$ arithmetic ensures consistently high throughput for edge processing.

Regarding decompression (Table IV), *CoST* achieves an average speed of 0.066 $\mu$s/point. This lightweight decoding process ensures rapid trajectory reconstruction on devices.

[1] https://www.modelscope.cn/datasets/opentrace/WorldTrace

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

[3] https://www.nbsdc.cn/general/dataLinks/16666.11.nbsdc.iaTZ0vFX

[4] https://anonymous.4open.science/r/CoST-0D8B

TABLE II
OVERALL COMPARISON - COMPRESSION RATIO (LOWER IS BETTER)

| Dataset | Floating-point | | | | | | Trajectory | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Lossless | | Lossy | | | | Batch | Micro-batch | | | | Stream |
| | Chimp | Elf | SZ2 | Machete | Serf-Qt | Serf-Xor | DP | OPW | D.R. | SQ-E | VOLT. | ***CoST*** |
| Geolife | 0.592 | 0.258 | 0.562 | 0.244 | 0.075 | 0.164 | 0.486 | 0.833 | 0.636 | 0.719 | 0.783 | **0.057** |
| WorldTrace | 0.640 | 0.558 | 0.407 | 0.220 | 0.084 | 0.143 | 0.271 | 0.503 | 0.323 | 0.423 | 0.372 | **0.057** |
| WX-Taxi | 0.553 | 0.313 | 0.769 | 0.296 | 0.147 | 0.174 | 0.676 | 0.954 | 0.713 | 0.818 | 0.821 | **0.130** |

TABLE III
OVERALL COMPARISON - COMPRESSION TIME ($\mu$S/POINT)

| Dataset | Floating-point | | | | | | Trajectory | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Lossless | | Lossy | | | | Batch | Micro-batch | | | | Stream |
| | Chimp | Elf | SZ2 | Machete | Serf-Qt | Serf-Xor | DP | OPW | D.R. | SQ-E | VOLT. | ***CoST*** |
| Geolife | 0.008 | 0.061 | 0.226 | 0.104 | **0.011** | 0.016 | 0.027 | 0.031 | 0.040 | 0.304 | 0.080 | 0.079 |
| WorldTrace | 0.007 | 0.063 | 0.763 | 0.052 | **0.009** | 0.012 | 0.011 | 0.365 | 0.039 | 0.079 | 0.075 | 0.047 |
| WX-Taxi | 0.010 | 0.066 | 0.275 | 0.112 | 0.013 | 0.017 | 0.027 | 0.339 | 0.031 | 0.197 | 0.078 | 0.235 |

TABLE IV
OVERALL COMPARISON - DECOMPRESSION TIME ($\mu$S/POINT)

| Dataset | Floating-point | | | | | | Trajectory | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Lossless | | Lossy | | | | Batch | Micro-batch | | | | Stream |
| | Chimp | Elf | SZ2 | Machete | Serf-Qt | Serf-Xor | DP | OPW | D.R. | SQ-E | VOLT. | ***CoST*** |
| Geolife | 0.010 | 0.011 | 0.106 | **0.001** | 0.015 | 0.008 | - | - | - | - | - | 0.051 |
| WorldTrace | 0.009 | 0.010 | 0.057 | **0.000** | 0.014 | 0.007 | - | - | - | - | - | 0.037 |
| WX-Taxi | 0.011 | 0.012 | 0.033 | 0.001 | 0.025 | 0.009 | - | - | - | - | - | 0.109 |

*3) Edge Latency Analysis:* In edge IoT environments, network bandwidth — rather than CPU capacity — often constitutes the primary bottleneck. We define the end-to-end latency for processing a single GPS point (original size 128 bits) as:

$$T_{total} = T_{comp} + T_{trans} + T_{decomp}. \tag{12}$$

where $T_{trans} = CompressedSize/Bandwidth$.

We simulate a constrained network at 500 kbit/s. Table V shows the latency breakdown on WorldTrace. Raw transmission alone costs 256 $\mu$s. Serf-Qt (CR $\approx 0.084$) reduces this to 21.63 $\mu$s (total $\approx 21.65$ $\mu$s). *CoST* (CR $\approx 0.057$) compresses to $\approx$ 7.28 bits, needing only 14.57 $\mu$s for transmission. Despite slightly higher computation (0.047 vs. 0.009 $\mu$s), it saves $> 7$ $\mu$s in transmission, reducing total latency by $\approx 32\%$ vs. Serf-Qt and $94\%$ vs. raw data.

### C. Discussion: Deployment Feasibility on Edge Devices

Although evaluated on a high-end CPU, *CoST* targets resource-constrained MCUs: it is $\mathcal{O}(1)$, uses **under 1 KB of RAM**, and with fixed $|S| = 3$ keeps predictor evaluation and adaptive-Huffman maintenance constant-time.

TABLE V
END-TO-END LATENCY COMPARISON ($\mu$S, 500 KBIT/S, WORLDTRACE)

| Algorithm | Comp. Time | Comp. Size (bits) | Trans. Time | Decomp. Time | Total Latency |
|---|---|---|---|---|---|
| Raw Data | **0.000** | 128.00 | 256.00 | **0.000** | 256.00 |
| Serf-Qt | 0.009 | 10.82 | 21.63 | 0.014 | 21.65 |
| Serf-Xor | 0.012 | 18.25 | 36.49 | 0.007 | 36.52 |
| ***CoST*** | 0.047 | **7.28** | **14.57** | 0.037 | **14.65** |

Even if an MCU were $1,000\times$ slower than our testbed ($T_{comp} \approx 50\mu s$), transmission would still dominate: at 500 kbit/s, raw points require $256\mu s$ while *CoST* needs only $14.6\mu s$, yielding over $190\mu s$ net latency reduction.

Currently, *CoST* assumes reliable in-order delivery; packet-loss recovery and robustness to severe GPS outliers are left for future work.

### D. Ablation Study

To quantify the contribution of each component, we evaluated five variants of *CoST* (Fig. 7): (1) **ZP / LP / CP**

(Single-Mode variants); (2) **Multi-3 (Switched)** (Brute-force approach); and (3) **Adaptive (*CoST*)** (The proposed cost-driven strategy).

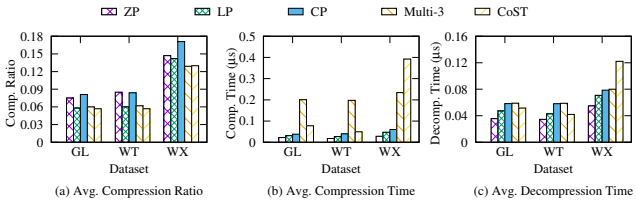


Fig. 7. Ablation Study Comparison.

Results indicate that single-mode predictors fail to generalize. Notably, "Linear Predictor (LP)" achieves the best single-mode CR on GeoLife (0.057) and WorldTrace (0.060) but struggles on WX-Taxi (0.142), whereas "Curve Predictor (CP)" generally introduces instability.

The Adaptive strategy (*CoST*) achieves the lowest compression ratio across all datasets. Regarding efficiency, *CoST* demonstrates smart adaptability. On GeoLife and WorldTrace, *CoST* ($0.078\mu s$, $0.050\mu s$) is faster than the Multi-3 brute-force approach ($0.201\mu s$, $0.198\mu s$) by avoiding unnecessary full-ensemble evaluations. On the complex WX-Taxi dataset, *CoST* incurs higher overhead ($0.393\mu s$ vs. $0.234\mu s$ for Multi-3) due to frequent mode switching and context updates, yet achieves the best compression performance.

### *E. Parameter Sensitivity*

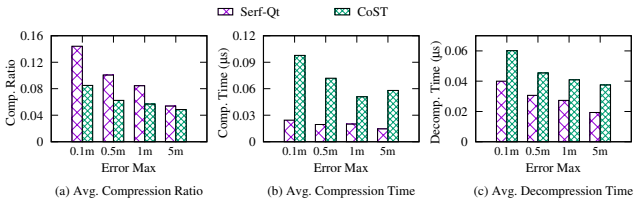


Fig. 8. Sensitivity Analysis of Error Bound.

*1) Error Bound:* Fig. 8 illustrates performance under varying error bounds. *CoST* consistently outperforms Serf-Qt in the primary range 0.1–5 m. At 1 m, *CoST* achieves CR 0.057 vs. Serf-Qt's 0.084. At extreme bounds (e.g., 30 m), Serf-Qt gains a slight edge (0.038 vs. 0.042) as the reduced signal-to-noise ratio limits kinematic prediction, but for high-precision applications *CoST* remains superior.

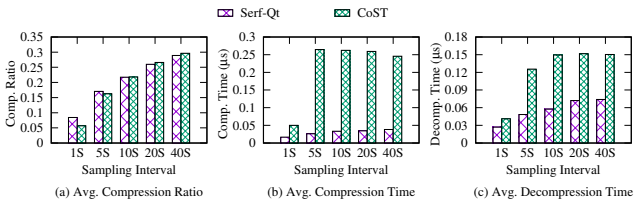


Fig. 9. Sensitivity Analysis of Sampling Rate.

*2) Sampling Rate:* Fig. 9 shows that *CoST* excels at high sampling rates (1s–5s). As the interval extends to 40s, its CR (0.296) converges towards Serf-Qt (0.289) because motion becomes less predictable; *CoST* is thus most effective for high-frequency streams.

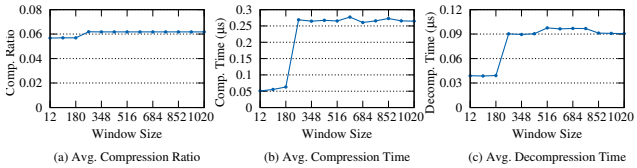


Fig. 10. Sensitivity Analysis of Window Size.

*3) Window Size:* Fig. 10 validates our choice of $W = 96$. Smaller sizes incur switching overheads, while sizes exceeding 200 sharply increase compression time (e.g., $\approx 0.27\mu s$ at $W = 264$) due to degraded cache locality. We fix $\mathcal{C}_{switch} = 1$ (a single mode bit); performance is far more sensitive to $W$ than to this constant. Auto-tuning $W$ is left for future work.

## VI. Conclusion

This paper presents *CoST*, a streaming trajectory compression framework specifically engineered for bandwidth-constrained edge environments. Addressing the limitations of existing batch-based and generic floating-point compressors, we introduce a strictly one-pass approach that effectively balances compression ratio, precision, and real-time latency.

The core innovation of *CoST* lies in its adaptive **Dual-Mode Prediction** mechanism, which dynamically switches between a lightweight **Linear Predictor (LP) Mode** for stable motion and a high-precision **Multi-Predictor (MP) Mode** for complex maneuvers. This adaptability is empowered by a novel **Parallel Cost Evaluation** strategy, ensuring optimal encoding decisions without pipeline stalls. Furthermore, by explicitly modeling spatio-temporal correlations under a principled physical error bound, *CoST* overcomes the theoretical barriers associated with treating spatial coordinates as independent variables.

Extensive evaluations on massive real-world datasets, including **WorldTrace**, **GeoLife**, and **WX-Taxi**, validate the efficacy of this design. *CoST* achieves compression ratios of up to 17.5×, outperforming state-of-the-art streaming algorithms by 32.6% to 60%. Crucially, it delivers an ultra-low end-to-end latency of approximately 14.65 $\mu$s per point, establishing it as a viable solution for delay-sensitive applications in intelligent transportation systems and IoT networks.

*CoST* is best suited to high-frequency telemetry with reliable in-order delivery; sparse sampling, severe GPS outliers, adaptive parameter tuning, and packet-loss recovery remain open issues. Future work will study FPGA acceleration, noise-aware variants, and extensions to other sensor streams.

## REFERENCES


[1] Y. Zheng, "Trajectory data mining: an overview," *ACM Trans. Intell. Syst. Technol.*, vol. 6, no. 3, pp. 1–41, 2015.

[2] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang, "T-Drive: Driving directions based on taxi trajectories," in *Proc. 18th ACM SIGSPATIAL Int. Conf. Adv. Geogr. Inf. Syst.*, 2010, pp. 99–108.

[3] Y. Zheng, L. Capra, O. Wolfson, and H. Yang, "Urban computing: concepts, methodologies, and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 5, no. 3, pp. 1–55, 2014.

[4] J. Zhang, Y. Zheng, D. Qi, R. Li, X. Yi, and T. Li, "Predicting citywide crowd flows using deep spatio-temporal residual networks," *Artif. Intell.*, vol. 259, pp. 147–166, 2018.

[5] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining (KDD)*. ACM, 2011, pp. 316–324.

[6] Y. Zheng, F. Liu, and H.-P. Hsieh, "U-Air: When urban air quality inference meets big data," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining (KDD)*, 2013, pp. 1436–1444.

[7] Y. Zheng, X. Xie, and W.-Y. Ma, "Collaborative location and activity recommendations with GPS history data," in *Proc. 19th Int. Conf. World Wide Web (WWW)*. ACM, 2010, pp. 1029–1038.

[8] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica*, vol. 10, no. 2, pp. 112–122, 1973.

[9] J. Muckell, P. W. Olsen Jr, J.-H. Hwang, C. T. Lawson, and S. Ravi, "Compression of trajectory data: a comprehensive evaluation and new approach," *GeoInformatica*, vol. 18, no. 3, pp. 435–460, 2014.

[10] T. Pelkonen, S. Franklin, J. Teller, P. Cavallaro, Q. Huang, J. Meza, and K. Veeraraghavan, "Gorilla: A fast, scalable, in-memory time series database," *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1816–1827, 2015.

[11] R. Li, Z. Chen, R. Lu, X. Xu, G. Yang, C. Chen, J. Bao, and Y. Zheng, "Serf: Streaming error-bounded floating-point compression," *Proc. ACM Manag. Data*, vol. 3, no. 3, pp. 1–27, 2025.

[12] K. Zheng, Y. Zheng, X. Xie, and X. Zhou, "Reducing uncertainty of low-sampling-rate trajectories," in *Proc. IEEE Int. Conf. Data Eng. (ICDE)*. IEEE, 2012, pp. 27–38.

[13] X. Yi, Y. Zheng, J. Zhang, and T. Li, "ST-MVL: Filling missing values in geo-sensory time series data," in *Proc. 25th Int. Joint Conf. Artif. Intell. (IJCAI)*, 2016, pp. 2704–2710.

[14] S. Wang, Z. Bao, J. S. Culpepper, and G. Cong, "A survey on trajectory data management, analytics, and learning," *ACM Comput. Surv.*, vol. 54, no. 2, pp. 1–36, 2021.

[15] W. Chen, Y. Liang, Y. Zhu, Y. Chang, K. Luo, H. Wang, Y. Wang, and Y. Zheng, "Deep learning for trajectory data management and mining: A survey and beyond," *arXiv preprint arXiv:2403.14151*, 2024.

[16] S. Di and F. Cappello, "Fast error-bounded lossy HPC data compression with SZ," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*. IEEE, 2016, pp. 730–739.

[17] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 12, pp. 2674–2683, 2014.

[18] Y. Shi, X. Zou, X. Chen, S. Jin, D. Tao, C. Deng, Y. Chen, and W. Xia, "Machete: An efficient lossy floating-point compressor designed for time series databases," in *Proc. Data Compress. Conf. (DCC)*. IEEE, 2024, pp. 532–541.

[19] N. Meratnia and R. D. By, "Spatiotemporal compression techniques for moving point objects," in *Adv. Database Technol. – EDBT 2004*. Springer, 2004, pp. 765–782.

[20] M. Chen, M. Xu, and P. Franti, "A fast $o(n)$ multiresolution polygonal approximation algorithm for GPS trajectory simplification," *IEEE Trans. Image Process.*, vol. 21, no. 5, pp. 2770–2785, 2012.

[21] Y. Han, W. Sun, and B. Zheng, "COMPRESS: A comprehensive framework of trajectory compression in road networks," *ACM Trans. Database Syst.*, vol. 42, no. 2, pp. 1–49, 2017.

[22] R. Song, W. Sun, B. Zheng, and Y. Zheng, "PRESS: A novel framework of trajectory compression in road networks," *Proc. VLDB Endow.*, vol. 7, no. 9, pp. 661–672, 2014.

[23] K. Wu, B. Zheng, and W. Sun, "Pilot-c: Physics-informed low-distortion optimal trajectory compression," *arXiv preprint arXiv:2508.03730*, 2025.

[24] D. Zhang, Z. Chang, D. Yang, D. Li, K.-L. Tan, K. Chen, and G. Chen, "Squid: subtrajectory query in trillion-scale gps database," *The VLDB Journal*, vol. 32, no. 4, pp. 887–904, 2023.

[25] T. Li, R. Huang, L. Chen, C. S. Jensen, and T. B. Pedersen, "Compression of uncertain trajectories in road networks," *Proc. VLDB Endow.*, vol. 13, no. 7, pp. 1050–1063, 2020.

[26] D. Zhang, M. Ding, D. Yang, Y. Liu, J. Fan, and X. Zhou, "Trajectory simplification: an experimental study and quality analysis," *Proc. VLDB Endow.*, vol. 11, no. 9, pp. 934–946, 2018.

[27] C. Long, R. C.-W. Wong, and H. Jagadish, "Direction-preserving trajectory simplification," in *Proc. VLDB Endow.*, vol. 6, no. 10. VLDB Endowment, 2013, pp. 949–960.

[28] P. Liakos, K. Papakonstantinopoulou, and Y. Kotidis, "Chimp: Efficient lossless floating-point compression for time series databases," *Proc. VLDB Endow.*, vol. 15, no. 11, pp. 3058–3070, 2022.

[29] R. Li, Z. Li, Y. Wu, C. Chen, and Y. Zheng, "Elf: Erasing-based lossless floating-point compression," *Proc. VLDB Endow.*, vol. 16, no. 7, pp. 1763–1776, 2023.

[30] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "An online algorithm for segmenting time series," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*. IEEE, 2001, pp. 289–296.

[31] Z. Cai, Q. Dong, M. Shi, X. Su, L. Guo, and Z. Ding, "VOLTCom: A novel online trajectory compression method based on vector processing," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 12, pp. 14 982–14 993, 2023.

[32] S. A. Shaikh, H. Kitagawa, A. Matono, and K.-S. Kim, "TStream: A framework for real-time and scalable trajectory stream processing and analysis," in *Proc. 30th Int. Conf. Adv. Geogr. Inf. Syst. (SIGSPATIAL)*, 2022, pp. 1–4.

[33] H. Yin, H. Gao, B. Wang, S. Li, and J. Li, "Efficient trajectory compression and range query processing," *World Wide Web*, vol. 25, no. 2, pp. 753–777, 2022.

[34] J. Liu, K. Zhao, P. Sommer, S. Garg, T. Schmid, and W. Lee, "Bounded quadrant system: Error-bounded trajectory compression on the go," in *Proc. 15th IEEE Int. Conf. Mob. Data Manage. (MDM)*, vol. 1. IEEE, 2014, pp. 98–107.

[35] X. Lin, J. Jiang, S. Ma, Y. Zuo, and C. Hu, "One-pass trajectory simplification using the synchronous Euclidean distance," *VLDB J.*, vol. 28, pp. 841–864, 2019.

[36] A. Alan, C. Yilmaz, and V. Ayaydın, "Dead reckoning for vehicle tracking with a smartphone," *IEEE Sensors J.*, vol. 19, no. 13, pp. 5137–5147, 2019.

[37] Google, "Protocol Buffers Encoding," [Online]. Available: https://protobuf.dev/programming-guides/encoding/, 2001, accessed: 2025-12-09.

[38] P. Elias, "Universal codeword sets and representations of the integers," *IEEE Trans. Inf. Theory*, vol. 21, no. 2, pp. 194–203, 1975.

[39] Y. Zhu, J. J. Yu, X. Zhao, X. Zhou, L. Han, X. Wei, and Y. Liang, "UniTraj: Learning a universal trajectory foundation model from billion-scale worldwide traces," *Adv. Neural Inf. Process. Syst.*, vol. 38, 2025.

[40] Y. Zheng, H. Fu, X. Xie, W.-Y. Ma, and Q. Li, "GeoLife GPS trajectory dataset - user guide," [Online]. Available: https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/, July 2011, geolife GPS trajectories 1.1.