

Visualization-Oriented Progressive Time Series Transformation

XIN CHEN^{*}, Renmin University of China, China

LINGYU ZHANG^{*}, Shandong University, China

HUAIWEI BAO, Shandong University, China

WEI LU, Renmin University of China, China

EUGENE WU, Columbia University, USA

XIAOHUI YU, York University, Canada

YUNHAI WANG[†], Renmin University of China, China

Visual analysis of large time-series data often requires transformations over multivariate time series. Existing methods struggle to meet interactive response time requirements, relying on full transformations that incur high computation costs. We propose a visualization-oriented transformation system PIVOT that incrementally generates accurate visualizations by selectively transforming only essential data samples. At its core is a *transformation-aware query mechanism* that efficiently computes point-wise transformations by leveraging cached hierarchical data on the server. To support responsive interaction, we introduce a *pixel-based error-bound guarantee* that estimates the accuracy of intermediate visualizations without requiring a reference, enabling a balance between latency and visual fidelity. Experiments show that PIVOT achieves highly accurate visualizations with interactive response times, outperforming existing error-free methods by up to an order of magnitude on billion-scale datasets.

CCS Concepts: • **Information systems** → **Query optimization**; • **Human-centered computing** → **Interactive systems and tools**; *Information visualization*.

Additional Key Words and Phrases: Time series, interactive progressive visualization, transformation

ACM Reference Format:

Xin Chen, Lingyu Zhang, Huaiwei Bao, Wei Lu, Eugene Wu, Xiaohui Yu, and Yunhai Wang. 2025. Visualization-Oriented Progressive Time Series Transformation. *Proc. ACM Manag. Data* 3, 6 (SIGMOD), Article 376 (December 2025), 26 pages. <https://doi.org/10.1145/3769841>

1 Introduction

Time-series data have experienced rapid growth in recent years across diverse domains, including finance, transportation, and manufacturing. Typically collected at regular intervals, this large-scale data is often stored in cloud-hosted databases, enabling efficient access and scalable processing.

^{*} Xin Chen and Lingyu Zhang are joint first authors.

[†] Yunhai Wang is the corresponding author.

Authors' Contact Information: Xin Chen, Renmin University of China, Beijing, China, chenxin19961029@ruc.edu.cn; Lingyu Zhang, Shandong University, Qingdao, China, zhanglingyu@mail.sdu.edu.cn; Huaiwei Bao, Shandong University, Qingdao, China, bhuaiwei@gmail.com; Wei Lu, Renmin University of China, Beijing, China, lu-wei@ruc.edu.cn; Eugene Wu, ewu@cs.columbia.edu, Columbia University, New York City, New York, USA, ewu@cs.columbia.edu; Xiaohui Yu, xhyu@yorku.ca, York University, New York City, New York, Canada, xhyu@yorku.ca; Yunhai Wang, Renmin University of China, Beijing, China, wang.yh@ruc.edu.cn.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM 2836-6573/2025/12-ART376

<https://doi.org/10.1145/3769841>

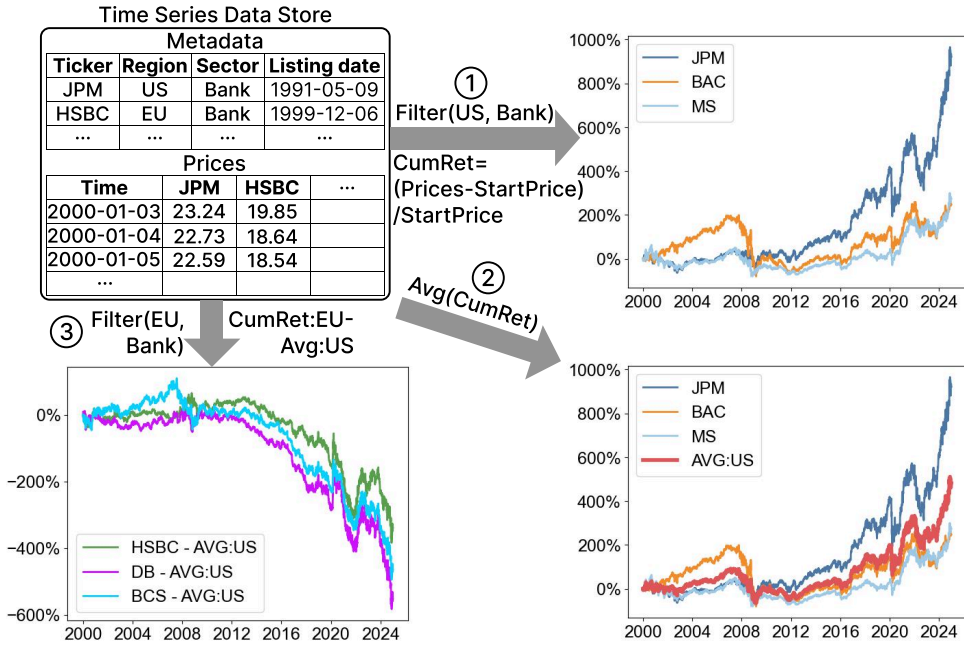


Fig. 1. Visual analysis of multiple time series from a dataset of NYSE-listed stocks. Analysts may casually apply and compose various point-wise transformations to explore interesting patterns, expecting timely and highly accurate visualizations.

Analyzing time-series data commonly involves applying various *point-wise transformations* [6, 8], which operate independently on values at each aligned timestamp. These transformations generally fall into two categories: (i) per-series operations, which modify individual time series using techniques such as the Box-Cox transformation or logarithmic scaling; and (ii) cross-series operations, which combine values from multiple aligned series—such as summation or subtraction at the same timestamp—to produce derived insights.

Figure 1 explores a collection of time series representing stock prices, each annotated with metadata such as region and sector. An analyst might begin by (1) filtering the series based on metadata predicates—for example, selecting bank stocks from the US—and then applying per-series point-wise transformations to compute cumulative returns relative to the initial price. Next, (2) the analyst may aggregate the transformed series by computing the average return across all selected US stocks for each timestamp. Finally, (3) this average could be subtracted from another group of series, such as EU stocks, to assess relative market performance. Such workflows often involve multiple transformation steps, with users interactively adjusting parameters and composing operators to uncover meaningful patterns. These types of analyses are *ad hoc* as part of the user’s data exploration, and so they are expected to respond quickly in order to maintain the user’s analysis flow [28].

As datasets grow larger, ensuring responsiveness for these *ad hoc* analyses is particularly challenging because results cannot be pre-computed. One approach involves first performing the necessary transformations on the server, followed by the application of visualization-driven aggregation techniques such as M4 [21] and OM³ [44]. These techniques help identify the essential records within each pixel column that preserve the exact rendering of the input time series. The selected records are then transmitted to the client for visualization, significantly reducing data transfer volume and enhancing rendering efficiency. However, executing this over the full dataset remains a bottleneck. For example, performing a logarithmic transformation to the *TLC Trip Record*

data [42] with 4.7 million data points takes 2.1 seconds on PostgreSQL and 1.2 seconds on DuckDB using a single thread. Alternatively, applying visualization-driven aggregation techniques first and then performing transformations on the selected records on the server can reduce overhead, but may compromise accuracy in the transformation results. This is because aggregation simplifies or approximates the data, potentially losing important details necessary for accurate visualizations.

To address this issue, a possible alternative is to adopt progressive visual analytics (PVA) [32, 41, 43, 46]. PVA delivers semantically meaningful partial results during data processing, allowing users to interact with evolving visualizations and adjust parameters in real time, without having to wait for the entire computation to complete. Most existing PVA techniques [13, 16, 30, 35], however, are primarily designed for approximate aggregate queries (e.g., SUM, COUNT, MIN) and are not well suited for point-wise data transformations. Even when adapted to support such transformations, these methods often require substantial computation over the full dataset to produce an accurate final visualization. Consequently, the overall computational cost remains high, limiting the practicality of existing PVA methods for large-scale time-series analysis.

In this paper, we introduce PIVOT for progressive visual analytics of massive time-series data that leverages visualization-aware optimizations to accurately visualize analysis results while significantly reducing computation costs compared to full data processing. PIVOT efficiently supports point-wise transformations while leveraging awareness of the visualization to integrate ideas from visualization-driven data aggregation, which helps users rapidly identify patterns of interest. Once such patterns are found, full transformations can be performed on the essential data to support deeper analysis. In doing so, PIVOT aligns with the natural workflow of visual exploration by prioritizing responsiveness and progressive refinement.

PIVOT builds on the *Time-series Aggregation Tree* (TAT), which hierarchically organizes time series to efficiently identify minimum and maximum values over arbitrary intervals by traversing only a subset of paths. Leveraging TAT, we propose a transformation-aware query mechanism that generates sufficiently accurate visualizations of transformation results, reducing redundant computation and query overhead. We show that for any point-wise transformation function $f()$ bounded over a given domain, a property satisfied by most transformations in visual analysis, the hierarchy can be aggressively pruned to rapidly respond to time-series interactions.

To further support user-driven analysis, we introduce a pixel-based error-bound guarantee that estimates the visual accuracy of intermediate results during TAT traversal in real time and enables more aggressive pruning. This enables users to dynamically balance response time and visual fidelity, refining transformations only when patterns or anomalies of interest are identified. Our system supports both per-series and cross-series operations, as well as core interactions such as zooming, panning, and dynamic transformation composition. Together, these capabilities offer a flexible and scalable foundation for the visual exploration of complex time-series data.

We evaluated PIVOT on time-series datasets with up to one billion records by quantitatively comparing it against a straightforward baseline: DuckDB [36] along with its inherently customized M4 [17], focusing on visual fidelity, processing time, and memory usage. The results show that PIVOT produces visualizations with user-controllable accuracy and achieves lower response times, even under error-free conditions, while also significantly reducing memory usage compared to DuckDB.

In summary, we make the following key contributions:

- We propose a novel system, PIVOT, a *Progressive Interactive Visualization-Oriented Transformation* system, for exploring large-scale time-series data interactively.
- We present a *transformation-aware query* that integrates sample selection, transformation, and visualization to accelerate computation while ensuring error-free visual output.

- We provide a *pixel-based error-bound guarantee*, enabling users to balance response time and visualization accuracy during interactive transformation.
- We quantitatively evaluate our method against state-of-the-art techniques in terms of both accuracy and efficiency.

2 Problem Formulation and Background

In this section, we first formally define the problem of visualization-oriented progressive time series transformation, followed by a brief overview of the relevant background.

2.1 Problem Formulation

We formalize two core problems related to visualizing transformation results over large-scale time series data: (1) efficiently generating error-free visualizations, and (2) generating approximate visualizations with guaranteed pixel-level error bounds.

Let $\mathbf{X} = \{X_1, \dots, X_m\}$ be a multivariate time-series dataset, where each time series X_i contains n elements, and $X_i[t]$ denotes the value of X_i at timestamp t . Interactive visual analysis often involves applying point-wise transformations to one or more time-series attributes, where the transformation is computed independently at each timestamp:

$$Y[t] = f(X_i[t], X_j[t], \dots, X_k[t]), \quad (1)$$

where X_i, X_j, \dots, X_k form a user-specified subset of \mathbf{X} , and the number of input variables l satisfies $l \in [1, m]$. Both the selection of input series and the temporal range over which Y is computed can be specified by the user.

A time series Y is often rendered as a line chart on a canvas of width w and height h , where each point $(t, Y[t])$ is mapped to canvas coordinates:

$$px(t) = w \cdot \frac{t - t_{\text{start}}}{t_{\text{end}} - t_{\text{start}}}, \quad (2)$$

$$py(Y[t]) = h \cdot \frac{Y[t] - Y_{\text{min}}}{Y_{\text{max}} - Y_{\text{min}}}. \quad (3)$$

Here, $t_{\text{start}}, t_{\text{end}}$ define the time range, and $Y_{\text{min}}, Y_{\text{max}}$ the value range. Following prior work [21, 44], we define the *rasterization* as the process of converting line segments between adjacent data points into a sequence of pixels. The pixels traversed by these segments are marked as foreground, and all others are considered background. Figure 2b shows an example where gray pixels represent the foreground corresponding to the time series in Figure 2a. However, not all data points contribute to the final rasterization, as traversing a pixel multiple times has the same visual effect as doing so once.

Definition 1. A pixel column on a canvas of width w corresponds to one of w equal-width intervals over the time range $[t_{\text{start}}, t_{\text{end}}]$. For the k -th interval, the data values in time series X_i form a group $C_{i,k} = \{X_i[t] \mid t \in [t_{\text{start}} + (k-1)\delta, t_{\text{start}} + k\delta]\}$, where $\delta \approx \frac{t_{\text{end}} - t_{\text{start}}}{w}$.

When the transformation function f is computationally expensive or the number of data points n is large, computing the full result Y as defined in Equation 1 can exceed acceptable latency for interaction, making it difficult for users to remain engaged. Inspired by the M4 approach [21], which demonstrates that rasterization within a pixel column relies only on a small set of key data points, we aim to progressively and efficiently generate error-free visualizations using a carefully selected subset of points from the original time series \mathbf{X} , particularly for common point-wise functions f .

Problem I (Error-free Visualization). Given a multivariate time series dataset \mathbf{X} and a transformation function $f : \mathbf{X} \rightarrow Y$, design a query mechanism Q that selects a subset $Q(\mathbf{X}) \subseteq \mathbf{X}$ such that the rendered visualization satisfies $V(f(Q(\mathbf{X}))) = V(f(\mathbf{X}))$ for any canvas resolution, where V denotes rasterization.

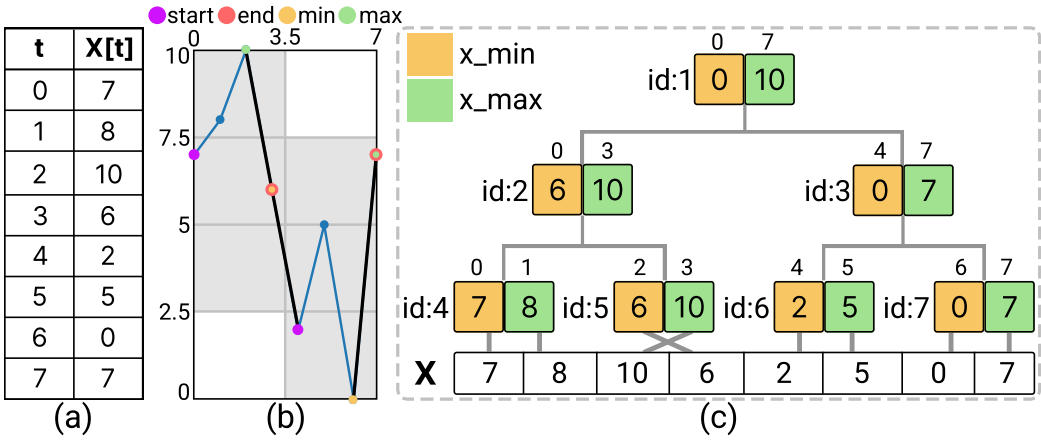


Fig. 2. Illustration of M4 and our proposed TAT representation with a sample time series in (a). (b) The corresponding line charts with two pixel columns: the blue line connects all data points, while the black line uses only M4-aggregated samples in each pixel column. (c) The corresponding TAT structure, where each node stores the minimum and maximum values and the associated time interval.

Although error-free visualizations are ideal, it is not always feasible under real-time constraints. When users seek early visual feedback during exploration, approximate visualizations with bounded perceptual error are acceptable [29]. This motivates a second problem formulation:

Problem II (Error-bounded Visualization). Given a multivariate time series dataset X , a function $f : X \rightarrow Y$, and a user-defined pixel error threshold τ , design a query mechanism Q that progressively refines a subset $Q(X)$ such that the pixel error rate upper bound ϵ of the visualization $V(f(Q(X)))$ satisfies $\epsilon \leq \tau$.

These two formulations capture the essential trade-off between accuracy and responsiveness in visualization-oriented time-series transformation. Our goal is to support both error-free and error-bounded modes, enabling interactive exploration of large-scale transformed time series with guaranteed visualization accuracy.

2.2 Background: M4 and Its Variants

For a given time series X_i , M4 [21] aggregation extracts the minimum, maximum, and the first and last data values within each pixel column. By connecting these four points in temporal order to form *inner-column line segments* and *inter-column line segments* and rasterizing these resulting segments, an error-free visualization can be produced using significantly fewer data points (see Figure 2b). However, retrieving the four aggregated samples for each pixel column requires scanning the entire dataset, resulting in a query time complexity of $O(n)$. Processing millions of records may take over one second, which is beyond the latency limit [28] for interactive visual analysis.

OM³ and MinMaxCache. To support interactive progressive visualization, OM³ [44] observed that an error-free result can be achieved using all inter-column line segments and only the inter-column segments whose rasterized pixels are not already covered. Notably, inner-column and inter-column segments are not connected unless they share points. Based on this, OM³ pre-processes a time series into a multi-level min max coefficient tree via a forward transform, recursively computing aggregates and encoding their differences to preserve detail. The resulting hierarchy, about three-quarters the size of the original data, is stored on the server.

At runtime, a visualization-aware incremental query algorithm with time complexity $O(w \log n)$ (w is the canvas width) retrieves only the coefficients needed for reconstruction via inverse transform. For each pixel column, the server streams coefficients for four M4 samples to the client. The visualization is progressively refined during traversal, using a pruning strategy that skips inter-column segments whose pixels are already covered by inner-column ones.

To eliminate the need for preprocessing, MinMaxCache [29] uses an adaptive caching strategy. Instead of fixed-width intervals, it caches min and max aggregates at dynamically chosen granularities to approximate M4 samples. Upon query, it checks whether the cached aggregates satisfy a pixel-level error bound; if not, it retrieves the missing values and updates the cache. This enables error-bounded visualizations without prior computation.

However, M4 and its variants target raw time-series visualization. When applied to our setting that involves visualizing transformed time series, they require full transformation results before rendering. Consequently, users must wait for costly computations to complete before gaining any visual insight.

3 TAT: Time-series Aggregation Tree

To solve Problem I, we propose a time-series aggregation hierarchy, TAT, which supports progressive visualization and enables fluid interaction for point-wise transformations through an efficient pruning strategy. Before introducing TAT, we first examine an approach that extends M4 to identify essential data points required for transformation. While effective in certain cases, this method is limited to scenarios where the transformation function is monotonic.

3.1 M4-based Transformation Approach

For a given time series X_i and a display window with width w , M4 [21] groups X_i into w columns. When a monotonic function f is applied to all data points in the k -th pixel column, the relative order of the values is preserved or reversed. Specifically, if f is monotonically increasing, then for all $X_i[t] < X_i[t']$ in this column, we have $f(X_i[t]) < f(X_i[t'])$; if f is monotonically decreasing, the order is reversed. In either case, the minimum and maximum values of $f(X_i)$ in the pixel column can be directly derived from the minimum and maximum of the original series X_i . Likewise, the first and last values of $f(X_i)$ in each pixel column are determined by the first and last values of X_i . Therefore, the pixels rasterized in the visualization of $f(X_i)$ are entirely determined by the M4-aggregated samples of X_i .

However, this only holds for monotonic univariate functions, as illustrated in Figure 3a. For non-monotonic transformations, even a single time series can lead to incorrect visualizations. For example, consider the function $f(x) = x \cdot \sin(x)$, which reaches its maximum at $x = 8$ within the first pixel column $C_{1,1}$ (see Figure 3b). The M4 aggregation for $C_{1,1}$ do not capture this peak, resulting in an incorrect visual range of $[f(10), f(6)]$. This issue also arises when applying point-wise functions to multiple time series. For example, the point-wise difference $f(X_1, X_2) = X_1 - X_2$ yields a minimum function value $f(7, 4)$ that does not align with the minimum s_{min} computed from the extrema of X_1 and X_2 , as illustrated in Figure 3c. Therefore, using only M4 aggregates from individual series is insufficient.

While M4 variants such as OM³ and MinMaxCache improve query performance, the data samples they retrieve only guarantee visual fidelity for visualizations of time series resulting from monotonic univariate transformations, similar to M4.

3.2 TAT and Its Property

Rather than operating directly on raw time series, our method constructs a hierarchical data structure, called the time-series aggregation tree (TAT). This structure is specifically designed

to selectively retrieve only the data points necessary for accurately rasterizing the results of non-monotonic or multivariate transformations, thereby reducing computational overhead.

As illustrated in Figure 2c, a full TAT is a complete binary tree in which each node is associated with the following attributes:

- id : uniquely identifies node, indexed from top to bottom.
- $\langle t_{start}, t_{end} \rangle$: start/end timestamps of interval the node represents.
- $[x_{min}, x_{max}]$: min/max values observed in node's interval.

For example, the root node in Figure 2c has an id of 1, a time range of $\langle 0, 7 \rangle$ and the value range of $[0, 10]$. The bottom layer of TAT consists of the original time series, with each data point acting as a leaf node. Before constructing TAT, we pad the number of leaf nodes to the next power of two by appending null values. For any parent node p , its temporal and value ranges are recursively computed from its two child nodes c_l and c_r :

$$\begin{aligned} p.t_{start} &= \min(c_l.t_{start}, c_r.t_{start}), \\ p.t_{end} &= \max(c_l.t_{end}, c_r.t_{end}), \end{aligned} \quad (4)$$

$$\begin{aligned} p.x_{min} &= \min(c_l.x_{min}, c_r.x_{min}), \\ p.x_{max} &= \max(c_l.x_{max}, c_r.x_{max}). \end{aligned} \quad (5)$$

Since the time-series data is ordered, $p.t_{start}$ must be equal to $c_l.t_{start}$, and $p.t_{end}$ must be equal to $c_r.t_{end}$. In contrast, $p.x_{min}$ and $p.x_{max}$ are computed by taking the minimum and maximum values from both child nodes. For example, in Figure 2c, the maximum value of the root node coincides with that of its left child.

Theorem 1. For a function bounded within a given domain, the minimum value within the sub-domain of a parent node p is always less than or equal to the minimum values within the sub-domains of its child nodes, c_l and c_r . Similarly, the maximum value of p is always greater than or equal to the maximum values of c_l and c_r .

PROOF. A fundamental property of the TAT is the transitivity of minimum and maximum values between a parent node p and its child nodes c_l and c_r , as defined in Equation 5:

$$p.x_{min} \leq c_l.x_{min} \quad \text{and} \quad p.x_{min} \leq c_r.x_{min},$$

$$p.x_{max} \geq c_l.x_{max} \quad \text{and} \quad p.x_{max} \geq c_r.x_{max}.$$

For a bounded function f , there exists a minimum value $s_{min} = f(\hat{x})$ within the range $[p.x_{min}, p.x_{max}]$. If \hat{x} is within $[c_l.x_{min}, c_l.x_{max}]$, then s_{min} is also the minimum in this range. Also, since $[c_r.x_{min}, c_r.x_{max}] \subseteq [p.x_{min}, p.x_{max}]$, s_{min} must be less than or equal to the minimum value in $[c_r.x_{min}, c_r.x_{max}]$. Similarly, if \hat{x} is within $[c_r.x_{min}, c_r.x_{max}]$, then s_{min} is the minimum in this range and must be less than or equal to the minimum in $[c_l.x_{min}, c_l.x_{max}]$. The same relationship holds for the maximum value.

Thus, for any function f bounded within the domain of a TAT, the following transitive relationships hold:

$$f(p).s_{min} \leq f(c_l).s_{min} \quad \text{and} \quad f(p).s_{min} \leq f(c_r).s_{min},$$

$$f(p).s_{max} \geq f(c_l).s_{max} \quad \text{and} \quad f(p).s_{max} \geq f(c_r).s_{max}. \quad (6)$$

When the function f is applied to multiple aligned time series, p represents a set of nodes with an identical ID across different TATs, and the same applies to c_l and c_r . \square

Based on this theorem, nodes c_l and c_r can be pruned if applying f to node p does not yield the minimum or maximum value in the current pixel column. For example, in Figure 3d, the right child nodes with value ranges $[6, 10]$ and $[0, 1]$ are skipped because the left child yields function values of $7 - 4 = 3$ and $8 - (-3) = 11$, which exceed the possible function value range of the right child. Additionally, we leverage this theorem to compute the theoretical lower and upper bounds of each node, which guides the traversal order during query processing. If f is differentiable over the

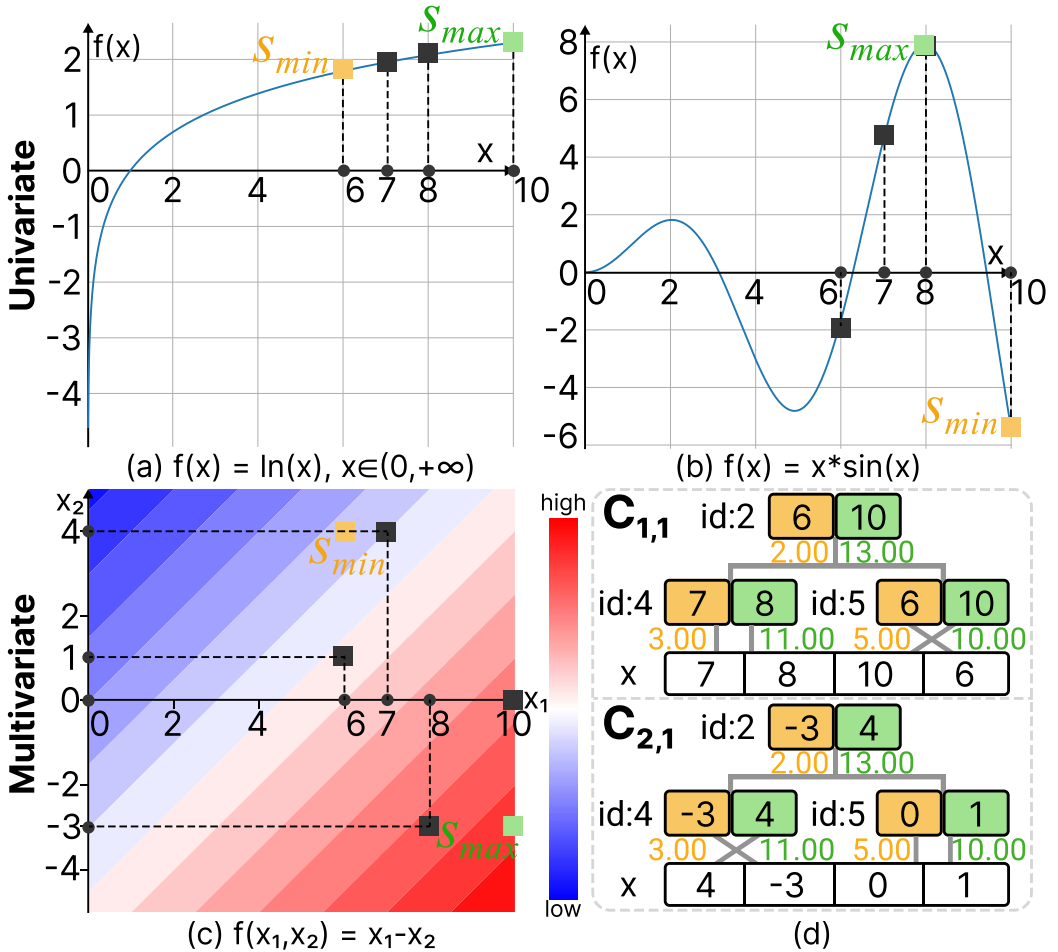


Fig. 3. Illustration of how our solution handles three types of transformation functions: (a) monotonic univariate, (b) non-monotonic univariate, and (c) bivariate. In all cases, the input time series has values $\{7, 8, 10, 6\}$, represented by black dots within a single pixel column. The corresponding function values are shown as black squares, while yellow and green squares mark the minimum and maximum function values within the domain, respectively. In (c), an additional time series $\{4, -3, 0, 1\}$ is used, and the corresponding TAT structures are illustrated in (d).

domain, its minimum and maximum values occur either at the domain boundaries or at stationary points where $f'(x) = 0$, according to *Fermat's Theorem* [1]. By precomputing the function values at these stationary points, we can reduce redundant evaluations of f during traversal, improving overall efficiency.

Yet, this theorem does not apply to unbounded functions, where it is infeasible to compute theoretical lower and upper bounds based on the closed interval defined by the node values x_{min} and x_{max} . For example, the function $f(x_1, x_2) = x_1/x_2$ approaches infinity at $x_2 = 0$, making the bounds undefined when $x_{min} < 0$ and $x_{max} > 0$.

While TAT can be dynamically constructed using range queries by using a node's start and end timestamps as query conditions and applying aggregation functions such as 'MIN()' and 'MAX()' to compute its value range, this approach presents several limitations. For unordered time-series data, each range scan incurs a time complexity of $O(n)$, resulting in inefficient construction.

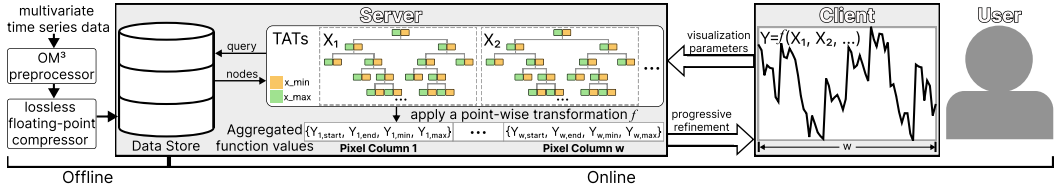


Fig. 4. Overview of our system PIVOT for exploring compressed large-scale time series stored in memory on a remote server.

Table 1. Notations

Symbol	Description
w, h	Width and height of the canvas
f	User-specified transformation function
\mathbf{X}	Input multivariate time series
X_i	The i -th univariate time series in \mathbf{X}
Y	Transformed result, i.e., $f(\mathbf{X})$
α	Extreme valid results observed in Y
β	Extreme theoretical scores computed over the input domain
γ	True extreme values in Y
η_i	Inner-column node with ID i
l_{min}, l_{max}	Candidate lists for minimum / maximum scores
ζ	Actual pixel error rate of an intermediate visualization
gR, R_k	Value range of the entire canvas / pixel column k
E_U	The union set of erroneous pixels under any gR and R_k
ϵ	Pixel error rate upper bound for given gR and R_k
τ	User-specified pixel error rate threshold

Moreover, it leads to redundant computation between parent and child nodes, as min and max values computed for child intervals are discarded and recomputed when constructing parent nodes. Fully materializing the tree structure also incurs significant storage overhead, requiring approximately $2n$ space.

4 PIVOT

In this section, we present PIVOT, a progressive interactive visualization-oriented transformation system. PIVOT enables real-time, user-driven exploration of large-scale multivariate time series through efficient point-wise transformations. Common notations used throughout this section are listed in Table 1.

4.1 Overview

PIVOT builds on top of TAT, a hierarchical structure constructed on the server at runtime. By avoiding repeated full data scans, PIVOT supports efficient reuse and accommodates dynamic updates of TATs during interaction. The system follows a client-server model, proceeding in two stages: *offline preprocessing* and *online querying* (see Figure 4).

Offline Preprocessing. We apply the OM^3 preprocessor to convert each time series into an ordered multi-level min-max coefficient tree via a forward transform. We reduce storage by compressing the detail coefficients with the ALP algorithm [2], which is also adopted by DuckDB [3].

Online Querying. PIVOT uses visualization parameters (e.g., the canvas width w , height h , time range $\langle t_{start}, t_{end} \rangle$, and user-specified analysis function f to reconstruct essential TAT nodes from the precomputed OM^3 coefficients and then applies the requested point-wise transformations on

the server. Once the necessary coefficients are retrieved, they are decompressed and inverted to recover the raw values via the inverse transform. PIVOT then applies point-wise transformations and aggregates the results by pixel column before progressively transmitting them to the client. To maintain interactive performance, PIVOT queries on-demand nodes incrementally, so that users can pan, zoom, resize, or otherwise explore the data seamlessly.

During query execution, PIVOT dynamically constructs partial TATs for fast access to relevant transformation samples. By performing transformation-aware, tree-based querying, PIVOT selectively retrieves TAT nodes likely to contain perceptually significant samples in each pixel column regarding function f . Each non-leaf node in TAT carries two *scores*: the theoretical minimum and maximum function values over its domain. Leaf nodes store the actual function values in Y . These scores guide the query to the next nodes of interest, while the scores and the leaf-node values collectively determine when the query process can stop. As discussed in the subsequent section, our transformation-aware query progressively gathers the aggregated function values $Y_{k,start}, Y_{k,end}, Y_{k,min}, Y_{k,max}$ needed for both error-free and error-bounded visualizations.

4.2 Transformation-aware Queries using TAT

For a display window of width w , the server processes the request by executing the transformation-aware query mechanism Q , which consists of three stages: identifying *boundary points* for each pixel column, *scoring nodes* within the current TAT structure, and performing a *depth-first traversal* to query and insert additional nodes as needed. The full procedure is detailed in Algorithm 1, with an illustrative example shown in Figure 5.

Identifying Boundary (lines 1-2). First, the server identifies TAT nodes corresponding to boundary timestamps of each pixel column k , using the canvas width w and time range $\langle t_{start}, t_{end} \rangle$:

$$t_{k,start} = t_{start} + k \cdot \delta, t_{k,end} = t_{start} + (k + 1) \cdot \delta - 1,$$

where $\delta = \frac{t_{end} - t_{start}}{w}$. Due to the full binary tree structure of the TAT, node IDs can be uniquely determined by these timestamps, making the detection of boundary nodes to be both deterministic and accurate. The relevant node IDs for these timestamps across all levels are combined into a single query, excluding the global t_{start} and t_{end} , as they do not contribute to the generation of inter-column lines. From the retrieved nodes, the server constructs a partial TAT for each time series corresponding to an attribute in the input multivariate dataset. Since both children of a parent node are reconstructed together, two types of nodes are distinguished: nodes that contain at least one boundary timestamp ($t_{k,start}$ or $t_{k,end}$) are defined as *boundary nodes*, while the remaining nodes that cover timestamps within column are referred to as *inner-column nodes*. A boundary node can be a leaf node, corresponding to a data point located at the boundary of pixel columns. For example, in Figure 5a, all retrieved nodes outlined in red are constructed during this stage. Among them, the red dashed boxes indicate boundary nodes and the lowest node with timestamp 21 is a leaf, corresponding to the last data point of pixel column C_1 .

By design, inner-column nodes obtained in this stage precisely cover the time intervals between the boundaries. As a result, the identified nodes are sufficient to derive not only $x_{k,start}$ and $x_{k,end}$ but also $x_{k,min}$ and $x_{k,max}$ for each pixel column. Together, they suffice for error-free visualizations of the input time series. However, unlike OM^3 , which prunes certain boundary nodes when their value ranges are fully contained within the overlap of adjacent pixel column ranges, we retain all boundary leaf nodes to ensure that all inter-column segments remain connected, as the applied transformation may cause their actual function value ranges to extend beyond those of the adjacent inner-column segments.

Scoring Nodes (lines 3-13). In this stage, for each boundary leaf node, we compute the function f directly to obtain valid results, which serve as the start and end values in the aggregated function

Algorithm 1 Transformation-aware Query Mechanism**Input:** Canvas width w , time range $\langle t_{start}, t_{end} \rangle$, dataset D , function f **Output:** M4 aggregation of transformation results Y

```

1:  $\mathcal{N} = \text{QueryDB}(D, \text{CalTimeRangeofAllCols}(w, t_{start}, t_{end}))$ 
2: TAT  $tr = \text{initializeTAT}(\mathcal{N})$ 
3: Initialize candidate node lists  $\{l_{k,min}\}_{k=1}^w, \{l_{k,max}\}_{k=1}^w$ 
4: for each pixel column  $k \in [1, w]$  do
5:    $[Y_{k,start}, Y_{k,end}, \alpha_{k,min}, \alpha_{k,max}] = \text{calFuncVals}(f, tr^k.\text{leaves})$ 
6:    $\mathcal{NS} = \text{calNodeScores}(f, tr^k.\text{innerColumnNodes})$ 
7:    $l_{k,min}.\text{heappush}(\mathcal{NS}), l_{k,max}.\text{heappush}(\mathcal{NS})$ 
8:    $\beta_{k,min} = l_{k,min}.\text{top}().\text{score}, \beta_{k,max} = l_{k,max}.\text{top}().\text{score}$ 
9: end for
10: if  $\text{IsUnivariateMonotonic}(f)$  then
11:   Set  $Y_{k,min}, Y_{k,max}$  by  $\beta_{k,min}, \beta_{k,max}$  for each column  $k$ 
12:   return  $Y$  ▷ Return exact results
13: end if
14:
15: while  $\text{CheckTermination}(\alpha, \beta)$  do
16:   for each pixel column  $k \in [1, w]$  do
17:      $id_{min} = l_{k,min}.\text{pop}().id, id_{max} = l_{k,max}.\text{pop}().id$ 
18:     while  $tr.\text{isNonLeafNode}(id_{min})$  do
19:        $[\eta_l, \eta_r] = tr.\text{split}(id_{min}, \text{QueryDB}(D, id_{min}))$ 
20:        $[ns_{large}, ns_{small}] = \text{calNodeScores}(f, \eta_l, \eta_r)$ 
21:        $id_{min} = ns_{small}.id$ 
22:        $l_{k,min}.\text{heappush}(ns_{large})$ 
23:     end while
24:     while  $tr.\text{isNonLeafNode}(id_{max})$  do
25:        $[\eta_l, \eta_r] = tr.\text{split}(id_{max}, \text{QueryDB}(D, id_{max}))$ 
26:        $[ns_{large}, ns_{small}] = \text{calNodeScores}(f, \eta_l, \eta_r)$ 
27:        $id_{max} = ns_{large}.id$ 
28:        $l_{k,max}.\text{heappush}(ns_{small})$ 
29:     end while
30:      $\alpha_{k,min} = \min(\text{calFuncVals}(f, tr.\text{val}(id_{min})), \alpha_{k,min})$ 
31:      $\alpha_{k,max} = \max(\text{calFuncVals}(f, tr.\text{val}(id_{max})), \alpha_{k,max})$ 
32:      $\beta_{k,min} = l_{k,min}.\text{top}().\text{score}, \beta_{k,max} = l_{k,max}.\text{top}().\text{score}$ 
33:      $Y_{k,min} = \alpha_{k,min}, Y_{k,max} = \alpha_{k,max}$ 
34:   end for
35:   yield  $Y$  ▷ Return progressive results
36: end while

```

values Y . These valid results initialize the minimum and maximum values of each pixel column k , denoted $\alpha_{k,min}$ and $\alpha_{k,max}$.

For every inner-column non-leaf node, we apply f to its domain to determine the node's minimum and maximum scores. Each scored node then goes into a candidate list, stored in a heap ordered first by score and then by node ID (see Figure 5b). In the corresponding minimum and maximum lists, the top entry in each denotes the most extreme scores, $\beta_{k,min}$ and $\beta_{k,max}$.

Based on these values, the true minimum and maximum of the transformed data, $\gamma_{k,min}$ and $\gamma_{k,max}$, must satisfy

$$\beta_{k,min} \leq \gamma_{k,min} \leq \alpha_{k,min} \text{ and } \alpha_{k,max} \leq \gamma_{k,max} \leq \beta_{k,max}. \quad (7)$$

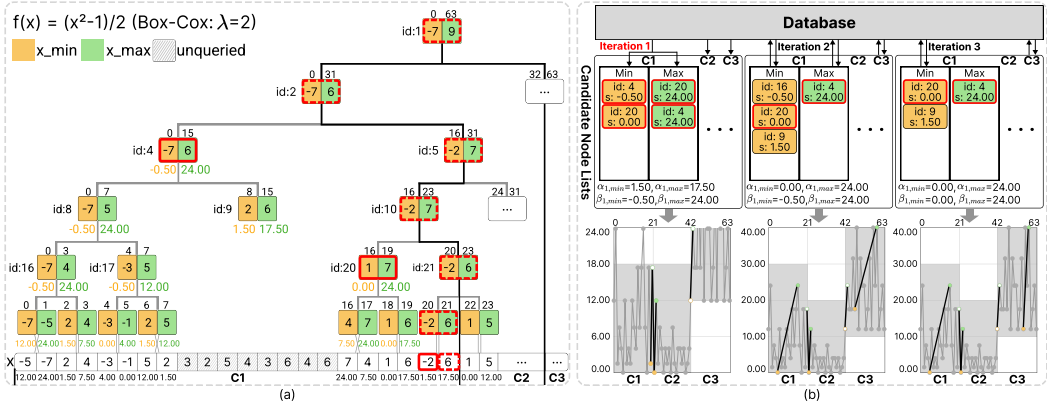


Fig. 5. Illustration of the query mechanism Q on a time series of length 64 over a canvas with three pixel columns. (a) For the non-monotonic function $f(x) = (x^2 - 1)/2$, Q retrieves aggregated values from the TAT, with node scores shown below. (b) Evolution of candidate lists α and β over the query process, alongside corresponding progressive visualizations. The ground truth $f(X)$, shown in gray, is for reference only; it is not computed during execution.

Here, $\alpha_{k,min}$ and $\alpha_{k,max}$ come from boundary leaf nodes that are already computed, and by Theorem 1, unqueried nodes cannot provide valid results below $\beta_{k,min}$ or above $\beta_{k,max}$.

For a univariate monotonic function, one can simply set $\gamma_{min} = \beta_{min}$ and $\gamma_{max} = \beta_{max}$ and thus obtain exact aggregated outcomes Y without querying additional data points. As an example, the Box-Cox transform [9] with $\lambda = 3$, $f_1(x) = (x^\lambda - 1)/\lambda$, can be applied directly to the M4-aggregated samples from the first pixel column in Figure 5a, $\eta_4 \cdot x_{min} = -7$, $\eta_4 \cdot x_{max} = 7$, $x_{1,end} = 6$, yielding $Y_{1,min} = -114$, $Y_{1,max} = 114$, $Y_{1,end} = 71.67$.

In contrast, for non-monotonic or multivariate functions that still satisfy Theorem 1, tree traversal is still required to retrieve additional nodes and iteratively update α and β , progressively refining the client-side visualization until it converges to the correct result. For instance, in the first pixel column of Figure 5a, the function values computed from the minimum value -7 and the maximum value 7 both yield 24.00 , which is insufficient to determine the true range of the transformation results; further traversal is therefore required. To minimize redundant computation and data transfer, TAT structures are cached on the server for reuse.

DFS Traversal (lines 15-36). To finalize the actual minimum and maximum function values $\gamma_{k,min}$ and $\gamma_{k,max}$ for each pixel column k , we perform a depth-first traversal guided by the current TATs and two lists (one for minimum, one for maximum). Consider the process of finding $\gamma_{k,min}$ as an example. In each iteration, we pop out the top node from the minimum list and query the database to reconstruct its two child nodes and compute their scores. If a child node has a higher minimum score or the same score but a smaller ID, it is inserted into the candidate list; otherwise, we continue expanding its child nodes until reaching a leaf, where valid results can be computed. Afterward, we update $\alpha_{k,min}$ and $\beta_{k,min}$. Once $\alpha_{k,min} \leq \beta_{k,min}$, the minimal aggregated result $\gamma_{k,min}$ is determined by $\alpha_{k,min}$, as $\beta_{k,min}$ is the smallest score in the candidate list and all un-queried nodes are descendants that cannot yield a lower score, according to Theorem 1. The same procedure is used to update $\alpha_{k,max}$ and $\beta_{k,max}$ to determine $\gamma_{k,max}$ for each pixel column. Throughout this process, we combine the valid results α_{min} and α_{max} across all columns with the Y_{start} and Y_{end} , to yield progressively transformation results. The query mechanism terminates until α and β for all pixel columns converge at γ , ensuring that the requirement in Problem I is met.

Figure 5 illustrates the evolution of the candidate node lists for the first pixel column C_1 during the query process. In the *Scoring Nodes* stage (Iteration 1), the function values of boundary leaf nodes are used to initialize $\alpha_{1,min} = 1.50$ and $\alpha_{1,max} = 17.50$, and we push the inner-column nodes η_4 and η_{20} into both lists. In Iteration 2, popping η_4 from the minimum list brings in η_4 , η_8 , and η_{17} are split to form a path to a new valid result, updating $\alpha_{1,min}$ to 0.00 at η_{34} and $\beta_{1,min}$ to -0.50 at η_{16} . For the maximum value, η_{20} is popped out due to its larger ID, causing $\alpha_{1,max}$ and $\beta_{1,max}$ to converge at $\gamma_{1,max} = 24.00$. Yet, determining $\gamma_{1,min}$ requires one more iteration to pop out and split η_{16} , ensuring that $\alpha_{1,min} \leq \beta_{1,min}$. Throughout the process, η_9 remains unsplit, leaving six nodes unqueried.

For the second pixel column C_2 , an extreme case arises where the minimum and maximum input values are 1 and 5. Since the scores of all inner-column nodes fall within the range $[f(1), f(5)] = [0.00, 12.00]$, the correct function value range can be determined without retrieving additional nodes, thereby significantly reducing response latency. The full example with all three pixel columns is provided in the appendix due to space constraints.

Algorithm 1 supports any point-wise transformation composition, including both per-series and cross-series operators. For example, the use case in Figure 1 defines a composite transformation $g(X_1, X_2, X_3, t)$, which comprises three component functions: cumulative return computation, averaging across multiple series, and point-wise subtraction. According to Equation 1, multiple input values aligned by the same timestamp are combined into a single output value. Hence, the query mechanism Q retrieves all required nodes with the identical ID across the involved time series and calculates a single pair of minimum and maximum scores. Therefore, auxiliary data structures like l_{min} and l_{max} remain applicable without modification, following the same usage described in Section 4.2 and the query process for $g(X_1, X_2, X_3, t)$ is shown in the appendix.

Acceleration Strategies. Although shown step by step for clarity, making a separate database query for each node expansion is expensive. We address this inefficiency in two ways. First, we batch the retrieval of child nodes by aggregating the IDs of all split candidates from l_{min} and l_{max} across pixel columns into a single query. Second, we balance performance and memory usage by using a three-level cache. Since the upper levels of the tree are traversed more frequently, the top 2^{12} nodes are both decompressed and decoded to allow fast access. The next 2^{20} nodes are decompressed into coefficients but remain undecoded, while the remaining nodes stay fully compressed in memory and are decompressed on demand. This caching design supports efficient in-memory processing of billion-scale time-series datasets.

Time Complexity. Given a canvas width w , the query process has a time complexity of $O(i \cdot w \cdot \log n)$, where n is the number of data points within the global time range, and i is the number of iterations, determined by the distribution of the input time series. In the worst case, all inner-column nodes may have scores higher than the actual extreme values γ , triggering full splits and resulting in a time complexity of $O(n)$. Although such cases are rare, they highlight the importance of incorporating an error-bound guarantee to balance visual accuracy and response latency.

4.3 Query with Pixel Error-bound Guarantees

In this section, we propose a pixel-based error-bound guarantee to enable users to dynamically balance interaction latency and visualization accuracy.

Pixel Error-bound Guarantees. The score-based DFS traversal naturally supports progressive refinement, gradually converging towards the actual extrema γ . However, it lacks a mechanism for early termination, as there is no metric to assess the accuracy of intermediate visualizations. For instance, in the first iteration of Figure 5b, $\alpha_{1,min}$ is initialized at 1.50, which overlaps with $\gamma_{1,min} = 0.00$ within the same pixel, yet this information is not visible to the user. To address this

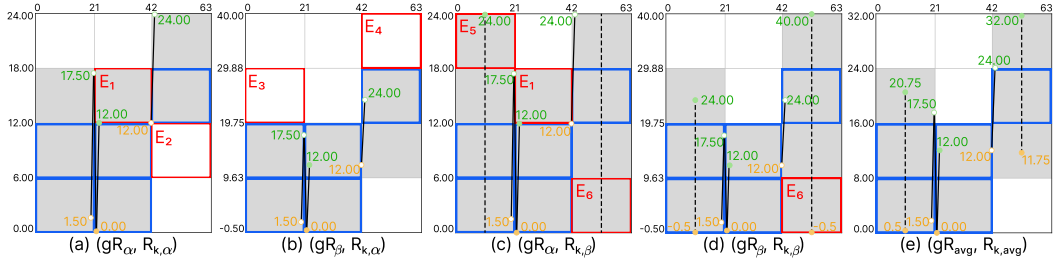


Fig. 6. Illustration of pixel errors under different combinations of R_k and gR . (a-d) The four extreme rasterization cases and (e) our average estimation approach using α and β values obtained in the first iteration. Red and blue boxes indicate erroneous pixels relative to the final visualization in Figure 5b and consistently rasterized pixels across all four cases, respectively. Black dashed lines serve as virtual guides for rasterization between R_k that do not correspond to valid results.

limitation, we propose an error-bound guarantee based on the α and β values across all pixel columns. Before introducing the guarantee, we first define the *actual pixel error rate* ζ as the ratio of the number of erroneous pixels, which differ from the error-free visualization, to the total number of pixels on the canvas.

Following MinMaxCache [29], our visualization-oriented error bound is defined as the *proportion of total pixels in the canvas that could be incorrect*, which differs from metrics that require the final visualization to be an exact reference image (e.g. SSIM [45]). Unlike MinMaxCache, where the input time series' global minimum and maximum values can be easily identified from the TAT, the global minimum and maximum function values remain unknown during the query process. Consequently, we must estimate both the value range for each pixel column, R_k , as well as the range of y -axis limits, gR . These estimates jointly determine how data points are mapped to the visualization and influence the pixel errors.

To render the resulting time series Y on the $w \times h$ canvas, the client can receive the pairs of $(\alpha_{k,min}, \alpha_{k,max})$ and $(\beta_{k,min}, \beta_{k,max})$ for each pixel column. According to Equation 7, the range gR must lie between the range of

$$gR_\alpha = [\min_{k=1}^w \alpha_{k,min}, \max_{k=1}^w \alpha_{k,max}] \quad (8)$$

and

$$gR_\beta = [\min_{k=1}^w \beta_{k,min}, \max_{k=1}^w \beta_{k,max}]. \quad (9)$$

Similarly, the range R_k of each pixel column must lie between $R_{k,\alpha} = [\alpha_{k,min}, \alpha_{k,max}]$ and $R_{k,\beta} = [\beta_{k,min}, \beta_{k,max}]$.

As a result, there are four possible extreme ways to rasterize the canvas, determined by different combinations of gR and R_k values: $(gR_\alpha, R_{k,\alpha})$, $(gR_\beta, R_{k,\alpha})$, $(gR_\alpha, R_{k,\beta})$, and $(gR_\beta, R_{k,\beta})$. Given that $\alpha_k \leq \gamma_k \leq \beta_k$, the cases $(gR_\alpha, R_{k,\beta})$ and $(gR_\beta, R_{k,\beta})$ may introduce false foreground pixels, while the case $(gR_\beta, R_{k,\alpha})$ may result in missing pixels. In contrast, the case $(gR_\alpha, R_{k,\alpha})$ can lead to both false and missing pixels, where gR and R_k are simultaneously compressed. Figure 6(a-d) shows an example, where α_k and β_k are obtained in the first iteration in Figure 5b. Taking the final visualization shown in the bottom right of Figure 5b as the ground truth, the erroneous pixels in each of the four cases are highlighted with red boxes for comparison.

During the progression of visual analysis, the accurate minimum and maximum values of each pixel column $\gamma_{k,min}$ and $\gamma_{k,max}$ remain unknown. Consequently, an accurate reference visualization is unavailable, and potential erroneous pixels are identified as the ones with high uncertainty, which are rasterized in some extreme cases but not others, as defined below.

Definition 2. The pixel errors E_U represent the difference between the union and intersection of the pixel ranges rasterized by four combinations of gR and R_k .

Theorem 2. E_U includes all erroneous pixels in the line chart visualization, ensuring that no errors occur outside E_U , regardless of the actual maximum and minimum function values γ_k .

PROOF. We separately analyze the erroneous pixels that occur when the values of gR or R_k are fixed, although gR depends on R_k .

When gR is fixed, only one variable y is present in Equation 3. Since γ_k lies between α_k and β_k , pixels rasterized in both cases of R_k , namely $R_{k,\alpha}$ and $R_{k,\beta}$, will also be rasterized for any value of γ_k . Therefore, the difference between the pixels rasterized by $R_{k,\alpha}$ and $R_{k,\beta}$ must cover all potential pixel errors when gR is fixed.

When the range values of R_k for each pixel column are fixed, Y_{min} and Y_{max} in Equation 3 are determined by gR and we have the following projection:

$$p_{\perp}(m, M) = h \cdot \frac{\hat{y} - m}{M - m},$$

where \hat{y} is one of range values of any R_k , and m and M represent gR_{min} and gR_{max} , implying $\hat{y} \geq m$ and $\hat{y} \leq M$. Differentiating this projection with respect to m and M gives:

$$\frac{\partial p_{\perp}}{\partial m} = h \cdot \frac{\hat{y} - M}{(M - m)^2}, \quad \frac{\partial p_{\perp}}{\partial M} = h \cdot \frac{-(\hat{y} - m)}{(M - m)^2},$$

which indicates that p_{\perp} is continuous and monotonic. Since gR must also lie between gR_{α} and gR_{β} , its projected pixels are in the ranges of the ones projected by using gR_{α} and gR_{β} . In other words, pixels rasterized in both gR_{α} and gR_{β} cases remain rasterized for any gR , and pixels rasterized in only one of these cases must cover all potential pixel errors when the range values of R_k are fixed.

Combining the above two scenarios, pixels rasterized in all four extreme cases will be rasterized in the final visualization, whereas the difference set E_U among these four cases covers all pixel errors, regardless of γ_k . \square

To illustrate this theorem, consider the third pixel column in the extreme cases shown in Figure 6a and 6c under a fixed global range (gR). This column includes several potential pixel errors, such as $\{E_2, E_6\}$, yet the top two pixels are always rasterized regardless of the value of γ_3 . In contrast, under a fixed column range (R_k), the third pixel column in Figure 6a and 6b consistently rasterizes the second pixel from the top and never the bottom one, regardless of how the y -axis range compresses or shifts the line segment [12, 24].

Based on Theorem 2, E_U in Figure 6 is obtained by taking the union of rasterized pixels excluding those that remain consistent (blue boxes) across all extreme cases. This result corresponds to the union of pixel errors highlighted by red boxes, confirming Theorem 2. Accordingly, we define *pixel error rate upper bound* as:

$$\varepsilon = \frac{|E_U|}{w \times h}. \quad (10)$$

By using a pixel error rate threshold τ to guide the query mechanism Q , a tunable trade-off between visual fidelity and response latency, as required in Problem II, is enabled. For example, in Figure 6, ε is 0.5, which exceeds the actual pixel error rates ζ in the four extreme cases: 1/6, 1/6, 1/4, and 1/12, respectively.

Query with Early Termination. To implement the error-bounded visualization (Problem II), we integrate the condition $\varepsilon \leq \tau$ into the $\text{CheckTermination}(\alpha, \beta)$ step (line 15) of Algorithm 1. Instead of checking the convergence of α_k and β_k for each pixel column k , ε , which depends on all pixel columns, is evaluated once per iteration. To optimize performance, we implement Definition 2 by calculating only the minimum and maximum endpoints of pixel ranges for each column under the four extreme cases, thus avoiding the need to rasterize the four visualizations. By differencing these

endpoints, we directly compute the number of potential pixel errors, $|E_U|$, as defined in Equation 10, without explicitly determining the error set E_U . Once ε falls below τ , the query process terminates. The default value for τ is set to 0.05, which provides a practical balance between accuracy and performance across the tested datasets.

In addition to providing strict error bounds, minimizing the actual error rate ζ in intermediate results is critical for effective progressive visualization [7]. A key challenge lies in estimating the value range of Y within each pixel column. Rather than using any of four combinations of gR and R_k values shown in Figure 6(a–d), we adopt the average estimators:

$$Y_{k,min} = (\alpha_{k,min} + \beta_{k,min})/2, Y_{k,max} = (\alpha_{k,max} + \beta_{k,max})/2.$$

as the mean generally provides a better estimate than the endpoints (α_k and β_k) for unknown distributions [10]. Accordingly, we update line 33 in Algorithm 1, so that the value ranges are computed as:

$$gR_{avg} = [\min_{k=1}^w Y_{k,min}, \max_{k=1}^w Y_{k,max}], R_{k,avg} = [Y_{k,min}, Y_{k,max}].$$

This modification helps the resulting line chart converge more quickly to an accurate visualization, as illustrated in Figure 6e. A comparison of the four combinations and average estimators across all tested datasets is provided in Figure 10c.

5 Evaluation

In this section, we evaluate the performance of PIVOT in terms of visualization accuracy, response time, and memory usage. We first describe the experimental setup, then compare it with competing methods in both cold-start and interaction scenarios under various parameter settings.

5.1 Experimental Setup

Competitors. We compare our system with a conventional pipeline that first performs transformations in the database, followed by visualization-driven aggregation using techniques like M4 to extract representative samples for rendering. Since our focus is on interactive analysis, all compressed coefficients are preloaded into server memory. Accordingly, we primarily compare against DuckDB [36], a state-of-the-art in-memory analytical database that uses the same compression algorithm as our system. To improve baseline performance, we incorporate AM4 [24], an optimized M4 variant tailored for analytical database architectures like DuckDB. We refer to this baseline as *DkM4*. We do not compare with the pipeline that first applies OM³ [44] before performing transformations, since it only guarantees correct visualizations for monotonic univariate functions, as discussed in Section 3.1. For comparison, we evaluate two versions of our system: the exact version (PIVOT) and an error-bounded version with $\tau = 5\%$, denoted as PIVOT-0.05.

Datasets. As summarized in Table 2, we use 16 datasets with diverse sizes and distributions, ranging from 5.26 million to 5.05 billion data points. Among the five real-world datasets, Power [20], Soccer [33], and Stock [39] have been used in prior studies such as M4 [21] and OM³ [44]. The Taxi [42] and Flow datasets come from the transportation domain, where point-wise transformations like computing transaction rates are common in visual analysis. Following the approach of Maroulis et al. [29], we generated 11 synthetic datasets using random walks, ranging from 5 million to 5 billion data points. Each dataset covers a 4-year time span, with sampling intervals adjusted to successively double the size of the previous dataset. We refer to this collection as *Syn5M–5B*.

Transformations. To ensure a comprehensive evaluation, we tested 13 representative transformations covering univariate, bivariate, and multivariate scenarios:

- *Univariate functions* include the monotonic function $\ln(x)$ and the non-monotonic function $g_1(x) = 0.001 \cdot x^3 - 3 \cdot x$.

Table 2. Dataset basic characteristics

Name	# Fields	# Data points
Flow	10	8,615,200
Power	7	226,733,542
Soccer	6	39,454,980
Stock	10	9,035,210
Taxi	11	52,066,080
Syn5M-5B	5	5.26M-5.05B

- *Bivariate functions* include the four basic arithmetic operations (+, −, ×, ÷) and the squared L_2 norm, defined as $L_2^2(x_1, x_2) = \sqrt{x_1^2 + x_2^2}$.
- *Multivariate functions* span a variety of common aggregations across all time series in the dataset: average (*avg*), variance (*var*), weighted sum (*wsum*), maximum (*max*), L_2 norm, and a more computation-intensive variant $L_2^{\ln}(\mathbf{X}) = \sqrt{\sum_{i=1}^m \ln(x_i + 1)^2}$.

Since $\ln(x)$ and the division operator (\div) are undefined for negative inputs, we filtered the datasets to ensure all values were strictly positive, as noted in Section 3.

Measures. Following prior work [21, 44], we evaluate both visualization quality and system performance using three measures:

- *Visual Quality:* Measured by the Structural Similarity Index (SSIM) [45], which quantifies the similarity between visualizations generated by each method and the ground truth (full data rendering). Higher values indicate better fidelity.
- *Response Time:* The total time from issuing a visualization request to rendering the result.
- *Memory Usage:* The peak memory consumed during query execution to generate the visualization, including the OM^3 coefficients, TAT, and other intermediate variables.

Hardware. The system follows a client-server architecture. The client, implemented in JavaScript, runs on Chrome (v131.0.6778.205) on a MacBook Pro with a dual-core 2.7 GHz processor and 8 GB of RAM. The middleware, developed in C++, runs on a server equipped with two 12-core Intel Xeon Silver 4410Y CPUs, 512 GB RAM, a 1 TB disk, and Ubuntu 20.04.

Scenarios. We evaluate system performance in two typical usage scenarios: cold-start and interaction. The cold-start scenario simulates the initial rendering of a static line chart based solely on user-defined parameters, without any cached data. In contrast, the interaction scenario reflects ongoing analysis, where cached results are reused to speed up frequent operations such as resizing, panning, zooming, and modifying variables or transformation functions.

5.2 Performance in Cold-start Scenarios

This section examines initialization overhead and how key parameters affect system performance in cold-start scenarios. Specifically, we analyze the influence of the transformation function, the number of input variables (l), the number of data points (n), the canvas width (w), as well as the error threshold (τ). To assess effectiveness, we compare our system against the DkM4 baseline under consistent conditions. Unless otherwise specified, the default settings are $w = 600$ and $h = 600$.

Initialization Overheads. We evaluated the preprocessing cost of PIVOT by measuring execution time and peak memory usage across all datasets. As expected, both metrics scale linearly with input size. For example, the smallest dataset (*Syn5M*) required 1.08 seconds and 0.08 GB of memory, while the largest (*Syn5B*) took 688.52 seconds (≈ 11 minutes) and 61.11 GB. Although preprocessing introduces additional cost and must be performed offline, this one-time overhead is amortized

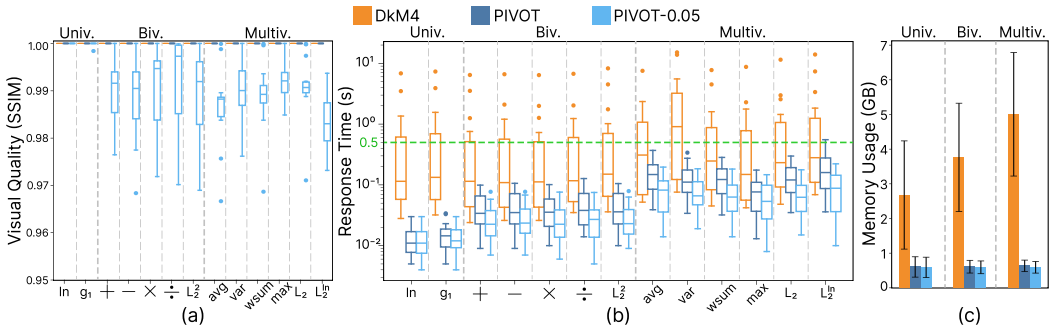


Fig. 7. Performance comparison of three systems using 13 representative transformation functions covering univariate, bivariate, and multivariate cases evaluated on all datasets: (a) SSIM, (b) response time, and (c) memory usage.

over repeated use and enables sub-second interaction latency. Detailed results for each dataset are provided in the appendix.

Types of Transformations. To assess how different types of transformation functions affect system performance, we evaluate 13 representative functions spanning univariate, bivariate, and multivariate cases. As shown in Figure 7, we report the visual quality, response time, and memory usage for each function.

The boxplots in Figure 7a summarize SSIM scores across all datasets. As expected, both DkM4 and PIVOT achieve perfect SSIM scores of 1.0 in all cases, as they are specifically designed to identify M4-aggregated samples of the transformation results. PIVOT-0.05 also achieves SSIM scores of 1.0 for all univariate functions, except for the piecewise monotonic function $g_1(x)$ on the *Flow* dataset, where the score slightly decreases to 0.998. For bivariate and multivariate functions, PIVOT-0.05 maintains median SSIM scores around 0.985, regardless of the number of input variables. This result aligns with expectations. For example, the monotonic function $\ln(x)$, discussed in Section 3.1, allows for precise identification of essential samples. Similarly, although $g_1(x)$ is only piecewise monotonic, accurate sampling is still possible when nodes fall within monotonic subranges. In contrast, bivariate and multivariate transformations depend on the pixel error threshold $\tau = 0.05$, producing approximate but consistently high-quality visualizations.

Figure 7b shows the response times for all transformation functions across the datasets. Both PIVOT and PIVOT-0.05 consistently remain below the interactive latency threshold of 0.5 seconds (green dashed line), while DkM4 often exceeds this limit, especially for bivariate and multivariate functions, and exhibits higher variability with more outliers. This aligns with our expectation that DkM4 incurs high latency on the largest datasets due to its linear time complexity, whereas PIVOT benefits from a tree-based query mechanism with logarithmic complexity. PIVOT-0.05 achieves the fastest performance due to its error-bounded strategy, which avoids computing exact values when unnecessary. On average, PIVOT and PIVOT-0.05 complete under 0.09 seconds and 0.05 seconds, whereas DkM4 takes 1.19 seconds. PIVOT achieves a mean speedup of 13.9 \times over DkM4, with PIVOT-0.05 improving further by 1.7 \times while maintaining high visual fidelity. This balance makes PIVOT-0.05 particularly suitable for exploratory analysis where speed is critical and slight visual approximations are acceptable.

Figure 7c presents the memory usage confidence intervals (CIs) for DkM4, PIVOT, and PIVOT-0.05 across all datasets across three categories of transformation. DkM4 consistently shows the highest memory consumption with noticeable variability and frequent outliers. In contrast, both PIVOT and PIVOT-0.05 maintain significantly lower and more stable memory usage, remaining well

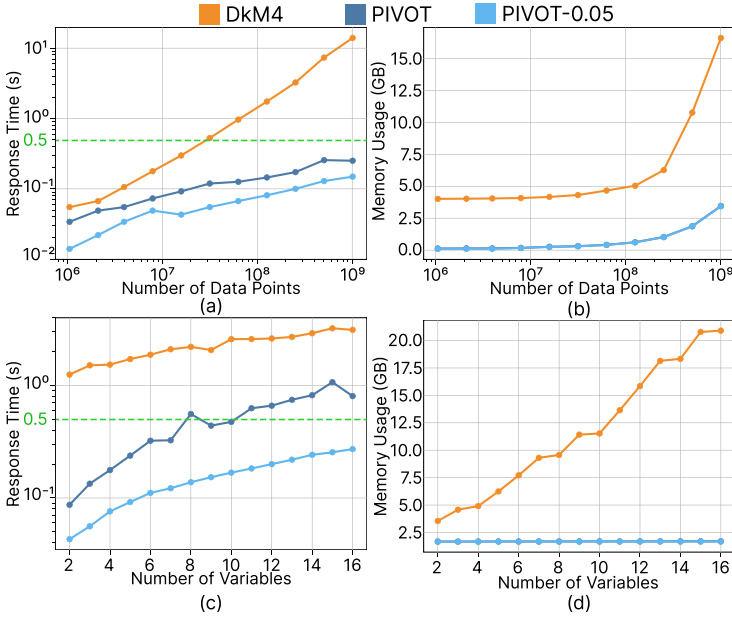


Fig. 8. Response time (a,c) and memory usage (b,d) of the three systems under varying (a,b) numbers of data points and (c,d) numbers of input variables when applying the transformation $L_2^{\text{In}}(X)$ to the synthesized time-series datasets.

below 1 GB across all transformation types. Specifically, PIVOT reduces memory usage by $5.21\times$ for univariate, $6.97\times$ for bivariate, and $9.03\times$ for multivariate transformations compared to DkM4. PIVOT-0.05 achieves slightly lower memory usage than PIVOT, although the difference is marginal. Upon careful inspection of the DuckDB configuration, we found that it applies compression only to persistent on-disk databases, not to in-memory instances [25]. In contrast, PIVOT stores data in compressed form in memory and decompresses them on demand, leading to significantly improved memory efficiency. These results highlight the efficiency of our method in handling large-scale time-series data with low memory overhead, making it particularly suitable for resource-constrained environments.

Number of Data Points and Variables. To evaluate how the number of data points n in Equation 1 affects system performance, we applied the most computationally intensive transformation, $L_2^{\text{In}}(X)$, to 11 synthetic datasets (*Syn5M–5B*) ranging from 5 million to 5 billion records, each with 5 fields. As shown in Figure 8a and Figure 8b, both response time and memory usage increase as the dataset size grows. DkM4 shows the highest and steepest increase in response time, surpassing 500 ms at 32M points and reaching 13.97 s at 5B points. In contrast, PIVOT maintains a stable response time of around 305 ms even for 5B points, while PIVOT-0.05 further reduces it to under 153 ms. All methods exhibit increased memory usage with larger datasets, particularly beyond 200M points. However, PIVOT and PIVOT-0.05 show nearly identical and significantly slower memory growth compared to DkM4. At 5B points, DkM4 consumes around 16.62 GB, while both PIVOT and PIVOT-0.05 use only about 3.45 GB. For smaller datasets (e.g., 5M points), DkM4 requires 4.02 GB, whereas PIVOT and PIVOT-0.05 use only 0.14 GB and 0.10 GB, respectively.

Similarly, we evaluate how the number of input variables l in Equation 1 affects system performance by synthesizing a time-series dataset with 128 million time points and 16 fields. We apply the transformation, $L_2^{\text{In}}(X)$, and progressively increase l from 2 to 16 by randomly selecting different fields. As shown in Figure 8c and Figure 8d, both response time and memory usage rise as more

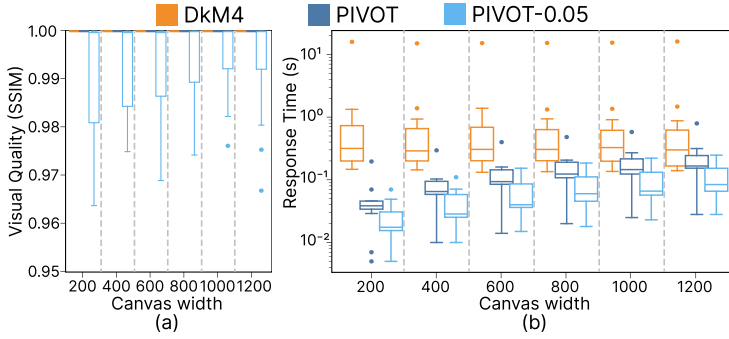


Fig. 9. Results of evaluation metrics on the largest real-world dataset *Power* as the canvas width varies: (a) SSIM scores and (b) response time.

variables are involved in the transformation. DkM4 exhibits the highest and most rapidly increasing response times, exceeding 1 second even for a small number of variables and approaching 3 seconds at $l = 16$. In contrast, PIVOT consistently outperforms DkM4, with response times increasing more gradually and remaining below approximately 1 second. PIVOT-0.05 achieves the best performance, maintaining latency under 300 ms across all settings. In terms of memory usage, DkM4 grows significantly with the number of variables, reaching 20.9 GB due to the overhead from SQL-based intermediate results. Conversely, PIVOT and PIVOT-0.05 maintain a stable memory footprint around 1.669 GB. Of this, approximately 1.666 GB is used to store compressed coefficients, and only a small fraction (0.002 GB) is needed for temporary node construction. As a result, both variants of our system demonstrate consistent memory efficiency regardless of the number of variables.

Canvas Width. We evaluate system performance on the largest real-world dataset, *Power*, using 13 transformation functions across six canvas widths ranging from 200 to 1200 pixels. Memory usage is omitted, as PIVOT and PIVOT-0.05 consistently consume significantly less memory than DkM4, as shown in Figure 8d.

Figure 9a shows that both DkM4 and PIVOT achieve perfect visual quality (SSIM = 1.0), while PIVOT-0.05 maintains high accuracy with median SSIM around 0.99 across all canvas widths. As illustrated in Figure 9b, response time increases with canvas width. At 1200 pixels, PIVOT maintains a median response time below 200 ms, though some outliers reach 793 ms. DkM4's median rises to around 300 ms, with outliers up to 16 s. In contrast, PIVOT-0.05 consistently achieves the lowest latency, with a median around 60 ms and all values under 250 ms.

Overall, PIVOT and PIVOT-0.05 deliver highly accurate visualizations while using significantly less memory than DkM4 across all types of transformations. Although response time increases moderately with transformation complexity and the number of variables, PIVOT-0.05 consistently maintains response times under 250 ms, even for transformations involving 1 billion data points and five variables.

Error Bound. We evaluate the effect of the pixel error-bound guarantee by running PIVOT with four user-specified thresholds of τ : 0% (error-free), 1%, 5%, and 10%. As shown in Figure 10a, average SSIM scores decrease with larger τ values, indicating a strong correlation between the threshold and perceived visual similarity. These results suggest that the proposed upper error bound ε (Equation 10) effectively quantifies visual quality.

Figure 10b illustrates the relationship between the pixel error rate upper bound ε (purple points) and the actual pixel error rate ζ (green points) under different τ settings (black horizontal lines), measured on the *Power* dataset. Results across 13 representative functions consistently show that

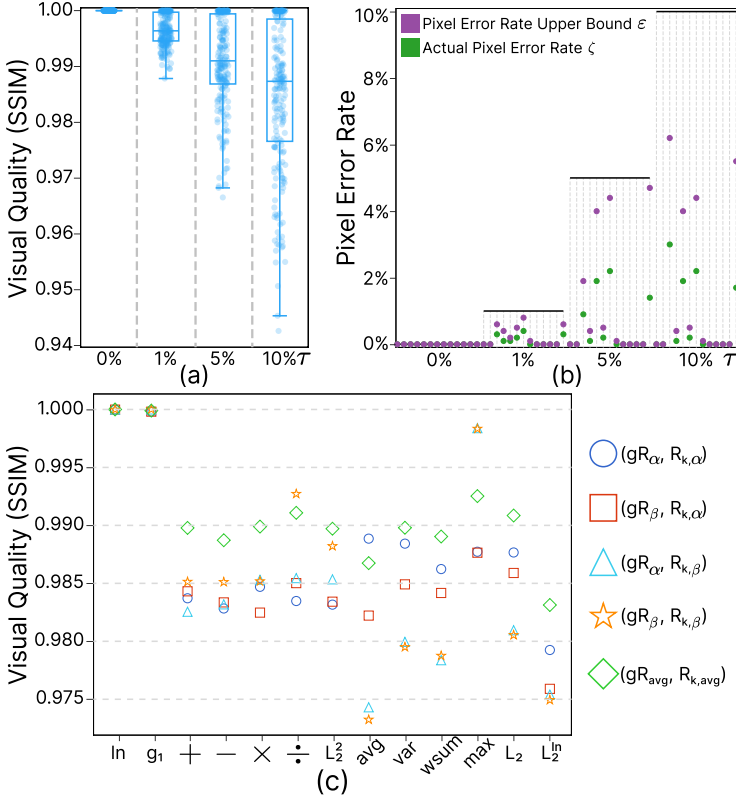


Fig. 10. Performance under varying pixel error rate thresholds τ . (a) The boxplots summarize the SSIM scores across all datasets. (b) The dot plot shows the relationship between pixel error rate upper bound ϵ and the actual pixel rate ζ on the *Power* dataset. (c) The dot plot displays the mean SSIM scores for various configurations with a fixed $\tau = 5\%$.

the actual error rate stays below the corresponding upper bound, confirming the effectiveness of our error-bound guarantee.

We further compare intermediate visualizations generated using either one of the four extreme value combinations or the average estimators (see Figure 6) under the default $\tau = 5\%$. As shown in Figure 10c, the average estimator achieves the highest mean SSIM scores in 8 out of 13 representative functions. For the remaining cases, all configurations perform nearly identically on the two univariate functions, while the average estimator closely approximates the best-performing configuration in the other three functions (\div , *avg*, *max*), with only slightly lower scores. These results demonstrate that the average estimator produces highly accurate intermediate visualizations for approximating the final output.

5.3 Performance in Interaction Scenarios

To simulate interactive exploration scenarios, we begin by applying the addition operation to the first two time series from one real-world dataset (*Taxi*) and one synthetic dataset (*Syn5B*) using an initial canvas size of 600×600 pixels. This is followed by a sequence of 50 interactions, each generated randomly according to a predefined query plan. Each interaction falls into one of the following three categories, selected with equal probability:

- *Transformation change*: a new transformation is randomly chosen from the 13 representative functions, ensuring it differs from the current one;

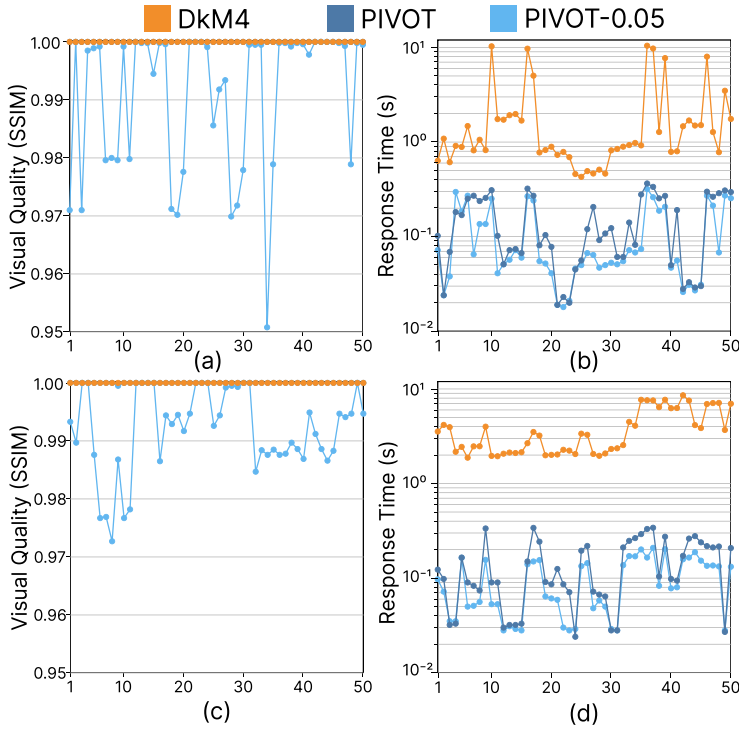


Fig. 11. Interactive performance on the real-world dataset *Taxi* (a, b) and synthetic dataset *Syn5B* (c, d): (a, c) SSIM scores; (b, d) response times.

- **Variable change:** a new combination of time-series fields is randomly selected from the dataset;
- **Visual interaction:** following the setup in [29], including *Panning* (shift the query window left or right by a random offset between 10% and 50% of the current time interval), *Zooming* (zoom in or out by a factor of 2), and *Resizing* (increase or decrease the canvas width by 50 pixels).

Each new interaction is executed immediately after the previous one completes, and the results are summarized in Figure 11. We omit memory usage here, as both of our systems maintain stable memory consumption, while DkM4 gradually increases over time; detailed memory usage results are provided in the appendix.

From Figure 11a, 11c, both DkM4 and PIVOT achieve perfect visual quality (SSIM = 1.0), while PIVOT-0.05 maintains consistently high accuracy, with most SSIM scores above 0.98 and occasional dips to 0.95 on the *Taxi* dataset. In terms of latency (Figure 11b, 11d), DkM4 exhibits the highest and most variable response times, often exceeding 1 second and peaking near 10 seconds. PIVOT reduces latency significantly, staying within 0.1–0.4 seconds, while PIVOT-0.05 performs best, consistently completing interactions in under 0.2 seconds and many within 0.05 seconds.

Comparing the two datasets, the synthetic data yields more stable results overall. PIVOT-0.05 maintains SSIM above 0.98 throughout on synthetic data, while on the *Taxi* dataset, SSIM occasionally dips to 0.95. Response times are also lower on the synthetic dataset for all methods: DkM4 stays below 8 seconds, PIVOT under 0.25 seconds, and PIVOT-0.05 under 0.15 seconds. On the real-world *Taxi* dataset, response times increase slightly, likely due to greater data complexity and noise.

In summary, both PIVOT and PIVOT-0.05 provide consistently low-latency and high-quality visualizations, with PIVOT-0.05 offering the fastest performance. While DkM4 struggles with higher latency, especially on real-world data, our systems remain robust across diverse interaction scenarios.

6 Related Work

Time-Series Transformation. A fundamental step in the exploratory analysis of time-series data is point-wise transformation [47], commonly used to normalize values or deriving new metrics through custom formulas. These operations can be applied to individual series (e.g., Box-Cox transformation, z-score normalization) or across multiple aligned series (e.g., point-wise subtraction or multiplication). Although widely adopted in commercial tools and research systems [4, 34, 36, 47], most existing approaches perform these transformations entirely on the server side before visualization. This design introduces two key limitations: (i) it requires complete computation of the transformed result before users can begin interacting with it, which delays feedback; and (ii) it lacks support for incremental refinement or early visual updates, making it difficult to support responsive, exploratory workflows. As time-series datasets continue to grow and are increasingly stored on remote or cloud-based platforms, these limitations become more pronounced, impeding real-time analysis. In contrast, our work introduces a visualization-oriented approach that tightly integrates transformation and visualization. Instead of waiting for the complete transformation, we enable progressive visualization by selecting only the most informative data samples for early visualization, significantly improving both the responsiveness and scalability of time-series exploration.

Progressive Visual Analytics. Progressive visual analytics (PVA) [11, 32, 41, 46] scales interactive systems by showing meaningful partial results during computation, allowing users to refine analysis in real time. Systems such as Falcon [31] and Sample+Seek [13] progressively render visual outputs as data becomes available. Some PVA systems also support time-series data [15, 19, 26, 40], but their capabilities are often limited to basic statistical summaries.

Sampling-based PVA methods [4, 13, 18, 23, 37] provide error-bound guarantees for aggregate queries (e.g., SUM, COUNT, AVG), enabling approximate yet reliable visualizations. For example, Kim et al. [23] proposed a rapid sampling algorithm for generating bar chart visualizations with probabilistic guarantees on the correctness of ordering, which is a critical visual property. However, these methods are not designed for point-wise transformations—operations applied to individual values at aligned timestamps—and typically overlook perceptual fidelity in the visual output. Although Plato [27] offers deterministic error guarantees for approximate analytics on compressed time series, its metrics (e.g., L_2 -norm) are disconnected from visualization quality. To bridge this gap, our work integrates transformation-aware sampling with progressive visualization, introducing a pixel-based error-bound guarantee specifically designed for time-series transformations. This enables responsive, high-fidelity visual analysis with interactive latency, while accounting for both computational efficiency and perceptual accuracy.

Visualization-driven Time Series Reduction. The rapid growth of time-series data has driven a few reduction techniques aimed at improving query efficiency and minimizing data size. Traditional methods [5, 12, 22] reduce dimensionality but often distort visual outputs, while line simplification techniques [14, 37] preserve shape but not perceptual fidelity (e.g., SSIM [45]). Visualization-driven approaches like M4 [21] preserve pixel-accurate charts by selecting key points per pixel column, but suffer from high query costs and lack reuse strategies. Recent systems such as OM³[44], MinMaxCache[29], and M4-LSM [38] improve performance via hierarchical indexing, caching, or log-structured storage, but require precomputed results and assume static transformations. Our method complements these efforts by enabling progressive, on-the-fly transformation of time-series data with pixel-based error-bound guarantees, supporting interactive exploration with both real-time responsiveness and transformation flexibility [47].

7 Conclusion

In this paper, we presented PIVOT, a progressive transformation system for interactive visual analysis of multivariate time series. Unlike traditional pipelines that separate visualization-driven aggregation and point-wise transformation into two distinct stages, PIVOT employs a transformation-aware query mechanism on time-series aggregation trees (TAT) to directly identify essential data samples for progressive refinement toward an error-free visualization. We further introduced a pixel-based error-bound guarantee that lets users balance visual accuracy and response latency. Experiments show that PIVOT outperforms existing methods with up to a 10× speedup.

Despite its advantages, PIVOT has limitations. It currently supports only point-wise transformations and relies on OM³'s offline preprocessing, making it unsuitable for streaming data. Future work includes extending support to multi-timestamp operations (e.g., cross-correlation), constructing TATs using dynamic caching strategies such as MinMaxCache to eliminate the need for preprocessing, and evaluating system performance under realistic cross-series workloads.

Acknowledgments

This work is supported by the grants of the National Key R&D Program of China under Grant 2022ZD0160805, NSFC (No.62132017 and No.U2436209), the Shandong Provincial Natural Science Foundation (No.ZQ2022JQ32), the Beijing Natural Science Foundation (L247027), the Fundamental Research Funds for the Central Universities, the Research Funds of Renmin University of China, and Big Data and Responsible Artificial Intelligence for National Governance, Renmin University of China.

References

- [1] Robert A Adams and Christopher Essex. 2018. *Calculus: A Complete Course*. Pearson.
- [2] Azim Afroozeh, Leonardo X Kuffo, and Peter Boncz. 2023. ALP: Adaptive Lossless Floating-Point Compression. *Proceedings of the ACM on Management of Data* 1, 4 (2023), 1–26.
- [3] Azim Afroozeh, Leonardo X Kuffo, and Sven Hepkema. 2024. ALP in DuckDB. <https://github.com/cwida/ALP>
- [4] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *Proceedings of the 8th ACM European Conference on Computer Systems (Prague, Czech Republic) (EuroSys '13)*. Association for Computing Machinery, New York, NY, USA, 29–42. doi:10.1145/2465351.2465355
- [5] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. 1993. Efficient Similarity Search in Sequence Databases. In *Foundations of Data Organization and Algorithms: 4th International Conference, FODO'93 Chicago, Illinois, USA, October 13–15, 1993 Proceedings 4*. Springer, 69–84.
- [6] Wolfgang Aigner, Silvia Miksch, Heidrun Schumann, and Christian Tominski. 2011. *Visualization of time-oriented data*. Vol. 4. Springer.
- [7] Marco Angelini, Giuseppe Santucci, Heidrun Schumann, and Hans-Jörg Schulz. 2018. A Review and Characterization of Progressive Visual Analytics. *Informatics* 5, 3 (2018). doi:10.3390/informatics5030031
- [8] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. 2015. *Time series analysis: forecasting and control*. John Wiley & Sons, Hoboken, NJ.
- [9] G. E. P. Box and D. R. Cox. 1964. An Analysis of Transformations. *Journal of the Royal Statistical Society: Series B (Methodological)* 26, 2 (1964), 211–243. doi:10.1111/j.2517-6161.1964.tb00553.x
- [10] George Casella and Roger Berger. 2024. *Statistical Inference. 2nd Edition*. CRC press, New York, NY, USA. doi:10.1201/9781003456285
- [11] Xin Chen, Jian Zhang, Chi-Wing Fu, Jean-Daniel Fekete, and Yunhai Wang. 2022. Pyramid-Based Scatterplots Sampling for Progressive and Streaming Data Visualization. *IEEE Transactions on Visualization and Computer Graphics* 28, 1 (Jan. 2022), 593–603. doi:10.1109/TVCG.2021.3114880
- [12] Graham Cormode, Minos Garofalakis, Peter J Haas, Chris Jermaine, et al. 2011. Synopses for Massive Data Samples, Histograms, Wavelets, Sketches. *Foundations and Trends® in Databases* 4, 1–3 (2011), 1–294.
- [13] Bolin Ding, Silu Huang, Surajit Chaudhuri, Kaushik Chakrabarti, and Chi Wang. 2016. Sample + Seek: Approximating Aggregates with Distribution Precision Guarantee. In *Proceedings of the 2016 International Conference on Management of Data*. 679–694.

- [14] 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.
- [15] Michael Glueck, Azam Khan, and Daniel J. Wigdor. 2014. Dive in! Enabling Progressive Loading for Real-Time Navigation of Data Visualizations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. Association for Computing Machinery, New York, NY, USA, 561–570. doi:10.1145/2556288.2557195
- [16] Anna Gogolou, Theophanis Tsandilas, Karima Echihabi, Anastasia Bezerianos, and Themis Palpanas. 2020. Data Series Progressive Similarity Search with Probabilistic Quality Guarantees. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1857–1873.
- [17] Jeffrey Heer and Dominik Moritz. 2023. Mosaic: An Architecture for Scalable & Interoperable Data Views. *IEEE Transactions on Visualization and Computer Graphics* 30, 1 (2023), 436–446.
- [18] Marius Hogräfer and Hans-Jörg Schulz. 2024. Tailorable Sampling for Progressive Visual Analytics. *IEEE Transactions on Visualization and Computer Graphics* 30, 8 (2024), 4809–4824. doi:10.1109/TVCG.2023.3278084
- [19] Jean-François Im, Félix Giguère Villegas, and Michael J. McGuffin. 2013. VisReduce: Fast and Responsive Incremental Information Visualization of Large Datasets. In *2013 IEEE International Conference on Big Data*. 25–32. doi:10.1109/BigData.2013.6691710
- [20] Zbigniew Jerzak, Thomas Heinze, Matthias Fehr, Daniel Gröber, Raik Hartung, and Nenad Stojanovic. 2012. The DEBS 2012 Grand Challenge. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems (Berlin, Germany) (DEBS '12)*. Association for Computing Machinery, New York, NY, USA, 393–398. doi:10.1145/2335484.2335536
- [21] Uwe Jugel, Zbigniew Jerzak, Gregor Hackenbroich, and Volker Markl. 2014. M4: A Visualization-Oriented Time Series Data Aggregation. *Proc. VLDB Endow.* 7, 10 (June 2014), 797–808. doi:10.14778/2732951.2732953
- [22] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. 2001. Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. *Knowledge and information Systems* 3 (2001), 263–286.
- [23] Albert Kim, Eric Blais, Aditya Parameswaran, Piotr Indyk, Sam Madden, and Ronitt Rubinfeld. 2015. Rapid Sampling for Visualizations with Ordering Guarantees. *Proc. VLDB Endow.* 8, 5 (Jan. 2015), 521–532. doi:10.14778/2735479.2735485
- [24] André Kohn, Dominik Moritz, and Thomas Neumann. 2023. DashQL—Complete Analysis Workflows with SQL. *arXiv preprint arXiv:2306.03714* (2023).
- [25] DuckDB Labs. 2025. Storage Versions and Format. <https://duckdb.org/docs/1.2/internals/storage>
- [26] Jianping Kelvin Li and Kwan-Liu Ma. 2020. P5: Portable Progressive Parallel Processing Pipelines for Interactive Data Analysis and Visualization. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (Jan. 2020), 1151–1160. doi:10.1109/TVCG.2019.2934537
- [27] Chunbin Lin, Etienne Boursier, and Yannic Papakonstantinou. 2020. Plato: Approximate Analytics over Compressed TimeSeries with Tight Deterministic Error Guarantees. *Proc. VLDB Endow.* 13, 7 (March 2020), 1105–1118. doi:10.14778/3384345.3384357
- [28] Zhicheng Liu and Jeffrey Heer. 2014. The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (Dec. 2014), 2122–2131. doi:10.1109/TVCG.2014.2346452
- [29] Stavros Maroulis, Vassilis Stamatopoulos, George Papastefanatos, and Manolis Terrovitis. 2024. Visualization-Aware Time Series Min-Max Caching with Error Bound Guarantees. *Proc. VLDB Endow.* 17, 8 (May 2024), 2091–2103. doi:10.14778/3659437.3659460
- [30] Dominik Moritz, Danyel Fisher, Bolin Ding, and Chi Wang. 2017. Trust, but Verify: Optimistic Visualizations of Approximate Queries for Exploring Big Data. In *Proceedings of the CHI conference on human factors in computing systems*. 2904–2915.
- [31] Dominik Moritz, Bill Howe, and Jeffrey Heer. 2019. Falcon: Balancing Interactive Latency and Resolution Sensitivity for Scalable Linked Visualizations. In *Proceedings of the 2019 CHI conference on human factors in computing systems*. 1–11.
- [32] Thomas Mühlbacher, Harald Piringer, Samuel Gratzl, Michael Sedlmair, and Marc Streit. 2014. Opening the Black Box: Strategies for Increased User Involvement in Existing Algorithm Implementations. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 1643–1652.
- [33] Christopher Mutschler, Holger Ziekow, and Zbigniew Jerzak. 2013. The DEBS 2013 grand challenge. In *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems (Arlington, Texas, USA) (DEBS '13)*. Association for Computing Machinery, New York, NY, USA, 289–294. doi:10.1145/2488222.2488283
- [34] Syeda Noor Zehra Naqvi, Sofia Yfantidou, and Esteban Zimányi. 2017. Time Series Databases and InfluxDB. *Studienarbeit, Université Libre de Bruxelles* 12 (2017).
- [35] Nicola Pezzotti, Boudewijn PF Lelieveldt, Laurens Van Der Maaten, Thomas Höllt, Elmar Eisemann, and Anna Vilanova. 2016. Approximated and User Steerable tSNE for Progressive Visual Analytics. *IEEE Transactions on Visualization and Computer Graphics* 23, 7 (2016), 1739–1752.

- [36] Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB: an Embeddable Analytical Database. In *Proceedings of the 2019 international conference on management of data*. 1981–1984.
- [37] Sajjadur Rahman, Maryam Aliakbarpour, Ha Kyung Kong, Eric Blais, Karrie Karahalios, Aditya Parameswaran, and Ronitt Rubinfeld. 2017. I’ve Seen “Enough”: Incrementally Improving Visualizations to Support Rapid Decision Making. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1262–1273.
- [38] Lei Rui, Xiangdong Huang, Shaoxu Song, Yuyuan Kang, Chen Wang, and Jianmin Wang. 2024. Time Series Representation for Visualization in Apache IoTDB. *Proc. ACM Manag. Data* 2, 1 (March 2024), 35:1–35:26. doi:10.1145/3639290
- [39] Debashis Sahoo. 2022. Stock Market Data. <https://www.kaggle.com/datasets/debashis74017/stock-market-data-nifty-50-stocks-1-min-data>
- [40] Shilpika, Takanori Fujiwara, Naohisa Sakamoto, Jorji Nonaka, and Kwan-Liu Ma. 2022. A Visual Analytics Approach for Hardware System Monitoring with Streaming Functional Data Analysis. *IEEE Transactions on Visualization and Computer Graphics* 28, 6 (June 2022), 2338–2349. doi:10.1109/TVCG.2022.3165348
- [41] Charles D Stolper, Adam Perer, and David Gotz. 2014. Progressive visual analytics: User-driven visual exploration of in-progress analytics. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 1653–1662.
- [42] New York City Taxi and Limousine Commission. 2024. TLC Trip Record Data. <https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page>
- [43] Cagatay Turkay, Nicola Pezzotti, Carsten Binnig, Hendrik Strobel, Barbara Hammer, Daniel A Keim, Jean-Daniel Fekete, Themis Palpanas, Yunhai Wang, and Florin Rusu. 2018. Progressive Data Science: Potential and Challenges. *arXiv preprint arXiv:1812.08032* (2018).
- [44] Yunhai Wang, Yuchun Wang, Xin Chen, Yue Zhao, Fan Zhang, Eugene Wu, Chi-Wing Fu, and Xiaohui Yu. 2023. OM3: An Ordered Multi-level Min-Max Representation for Interactive Progressive Visualization of Time Series. *Proc. ACM Manag. Data* 1, 2, Article 145 (June 2023), 24 pages. doi:10.1145/3589290
- [45] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. 2004. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing* 13, 4 (2004), 600–612.
- [46] Emanuel Zraggen, Alex Galakatos, Andrew Crotty, Jean-Daniel Fekete, and Tim Kraska. 2016. How Progressive Visualizations Affect Exploratory Analysis. *IEEE Transactions on Visualization and Computer Graphics* 23, 8 (2016), 1977–1987.
- [47] Jian Zhao, Fanny Chevalier, Emmanuel Pietriga, and Ravin Balakrishnan. 2011. Exploratory Analysis of Time-Series with ChronoLenses. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (Dec. 2011), 2422–2431. doi:10.1109/TVCG.2011.195

Received April 2025; revised July 2025; accepted August 2025