Visualization-aware Time Series Min-Max Caching with Error Bound Guarantees

Stavros Maroulis
ATHENA Research Center
Athens, Greece
stavmars@athenarc.gr

Vassilis Stamatopoulos
ATHENA Research Center
Athens, Greece
bstam@athenarc.gr

George Papastefanatos
ATHENA Research Center
Athens, Greece
gpapas@athenarc.gr

Manolis Terrovitis
ATHENA Research Center
Athens, Greece
mter@athenarc.gr

ABSTRACT

This paper addresses the challenges in interactive visual exploration of large multi-variate time series data. Traditional data reduction techniques may improve latency but can distort visualizations. State-of-the-art methods aimed at 100% accurate visualization often fail to maintain interactive response times or require excessive preprocessing and additional storage. We propose an in-memory adaptive caching approach, MinMaxCache, that efficiently reuses previous query results to accelerate visualization performance within accuracy constraints. MinMaxCache fetches data at adaptively determined aggregation granularities to maintain interactive response times and generate approximate visualizations with accuracy guarantees. Our results show that it is up to 10 times faster than current solutions without significant accuracy compromise.

PVLDB Reference Format:
Stavros Maroulis, Vassilis Stamatopoulos, George Papastefanatos, and Manolis Terrovitis. Visualization-aware Time Series Min-Max Caching with Error Bound Guarantees. PVLDB, 17(8): 2091 - 2103, 2024.
doi:10.14778/3659437.3659460

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/athenarc/MinMaxCache.

1 INTRODUCTION

The visual analysis of time series data collected nowadays (e.g. from IoT devices), presents significant challenges due to the vast volume and multi-variate nature of the data. The latency associated with processing and fetching the data for visualization can hinder interactivity. Traditional approaches to reduce data and improve visualization latency, such as sampling or aggregating at coarser granularities, may distort the resulting time series visualization and compromise the accuracy of the final representation compared with the raw data visualization. The latter may affect the effectiveness of several exploration and visual analytic tasks, such as visual anomaly detection or multivariate correlation analysis.

State-of-the-art techniques [15, 36], aim at minimizing latency while ensuring 100% accurate visualization results. Visualization-aware data aggregation M4 [15] takes into account the visualization's dimensions and aggregates data in pixel-wide time intervals. However, M4 has a significant drawback, as shown in Figure 1 in which a user visualizes a time series and performs a pan right operation: *the data fetched cannot be cached to accelerate subsequent user interactions, such as panning and zooming*, which usually refer to overlapping time intervals. Such operations alter the time interval over which data are aggregated and mapped to pixels, and require to process and re-fetch the data from the datastore. As a result, a time overhead is incurred, which becomes a significant concern for interactive response times [23] in large time series datasets.

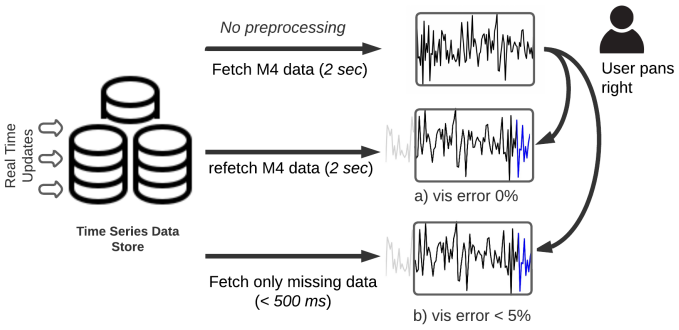


Figure 1: Visual Exploration of Time Series: a) Non-interactive, error-free visualization from M4 query over the complete interval, b) Interactive, error-bounded visualization reusing query results from prior operations.

To address these limitations, the recent work, OM3 [36] materializes a multi-level representation of the original time series and proposes incremental query evaluation and progressive visualization over it, eventually converging to an error-free result. Despite this, OM3 lacks any error guarantees for intermediate visualizations, forcing users to wait until the final output for any accuracy assurance. Further, OM3 requires *a very long pre-processing stage and additional storage space in the database to generate and store the multi-level representation of the time series*. These costs can be hard

Proceedings of the VLDB Endowment, Vol. 17, No. 8 ISSN 2150-8097.
doi:10.14778/3659437.3659460

to justify, considering that users often focus on recent data and OM3 does not support updates. Also, OM3 does not explicitly consider the multivariate nature of time series data collected from IoT devices, which often involve measurements of multiple variables.

Enhancing latency in time series querying can be accomplished with generic caching methods like TSCache [22], which offers interval-based caching of previous query results. However, such generic approaches do not consider the specific requirements of interactive visualization scenarios, where data are visualized in adjacent time intervals. Being agnostic on how cached data can be aggregated and reused for visualizing adjacent areas has impact on the overall performance and the accuracy of the visualization.

This work considers that visualized results can be cached and reused to approximate the rendering of subsequent user requests and accelerate interaction in a user exploration session. Such an approximation, however, may compromise the visualization quality with respect to how the actual raw data would be rendered. The bottom case in Figure 1 motivates this setting; previous visualized results are used to offer faster response time with a slight decrease (5%) in accuracy. This trade-off between speed and accuracy applies in domains with large data volumes that hinder interactive exploration. For example, in renewable energy parks, monitoring solar panels and wind turbines requires exploration over numerous sensor measurements, to spot performance degradation, or equipment malfunctioning. Providing accuracy guarantees remains crucial for the reliable assessment of such anomalies. Should analysts spot potential issues during exploration, they can then refine their error bounds to obtain an error-free visualization for verification.

In our approach, we assume that the error bound constraints are implicitly defined by the user. We identify the following key challenges: i) how do we dynamically *balance interaction time and visualization accuracy* in a multi-variate time series exploration setting? ii) how do we compute *error-bound guarantees* for the visualization accuracy? iii) how do we *avoid offline preprocessing* to support online analysis of continuously generated data?

To address these challenges, we present *MinMaxCache*, an in-memory adaptive caching approach that enables interactive visual exploration of large multivariate time series data while adhering to visualization accuracy constraints. Instead of caching data to pixel-wide intervals, our method caches min-max aggregates at intervals of dynamically determined granularities, which are then mapped to pixels to generate the visualization. Upon receiving a query, *MinMaxCache* leverages cached data to assess the visualization error bound and guarantee the query's accuracy requirements, while maintaining interactive response times. In the same time, it refines the granularity of the fetched data to enable reuse by future interactions. The key contributions of this work are:

- We define the problem of approximate visualization of time series with pixel error-bound guarantees, which offers a trade-off between accuracy and performance of interactivity in visual exploration of large time series datasets.
- We provide theoretical guarantees for an upper bound of pixel errors, when previously visualized results are cached and reused.
- We propose a solution for a visualization-aware caching mechanism and an approximate visualization method that adapts to user-specified error bounds, requires no preprocessing and provides interactive visualizations over large time series data.
- We evaluate our approach with real and synthetic data. Our results demonstrate our method is 6-10 times faster than competitors, significantly improving interactivity.

Outline. Section 2 provides an overview of the system, Section 3 presents background information and the error-bounded time series visualization problem. Section 4 describes the proposed cache structure and query evaluation methods. Section 5 presents the experimental evaluation and key insights from our user study, Section 6 reviews related work, and, finally, Section 7 concludes the paper.

2 SYSTEM OVERVIEW

Figure 2 presents an overview of our system. *MinMaxCache* acts as an intermediate layer between the visualization front-end and the underlying data store. Its role is to manage data retrieval and caching at appropriate granularities, thereby minimizing latency and maintaining an acceptable level of visualization accuracy.

Upon user interaction, the *Visualization Front-end* sends the visual operation and several parameters to MinMaxCache, including the time series dataset, the variables to be visualized, the requested time interval, and some visualization- and analysis-specific parameters (i.e., width and height of the chart, upper error bound). The *Visual Operation Evaluation Module* assesses the cached time series data to ascertain whether it can be used to generate the required visualization. The cache retains time series intervals and stores min-max aggregates for each, adaptively determining aggregation intervals to ensure visualization accuracy.

Based on the cached data, the system decides whether access to the data store is needed to fetch either missing or all the data of the query despite the interval being cached. This decision is dictated by the appropriateness of the aggregation granularity in guaranteeing the visualization accuracy, requested in the form of error bound by the application. In case a query to the underlying data store is required, the fetched results are integrated with the cached data to deliver the query results to the visualization front-end and the newly acquired data append the cache. The *Cache Manager* module also maintains the cache within its defined size limits, replacing less relevant data based on user operations.

MinMaxCache offers support for time series databases such as InfluxDB, SQL databases like PostgreSQL (details in Section 5), and a simple query interface (details in Section 4.2) that can easily communicate with several visualization libraries and tools[1].

3 ERROR-BOUNDED VISUALIZATION

In this section, we introduce our error-bounded time series visualization approach. First, we provide a brief overview of the main concepts and next we present our approach.

3.1 Preliminaries

Time series Data. Multivariate time series data is a sequence of n data points, i.e., $T = \{(t_1, V_1), (t_2, V_2), \ldots, (t_n, V_n)\}$. Each point has a timestamp t_i and a set of d numerical values $V_i = (v_{i,1}, v_{i,2}, \ldots, v_{i,d})$ corresponding to variables $Y = (y_1, y_2, \ldots, y_d)$ at t_i.

[1]Code and data are open sourced in https://github.com/athenarc/MinMaxCache

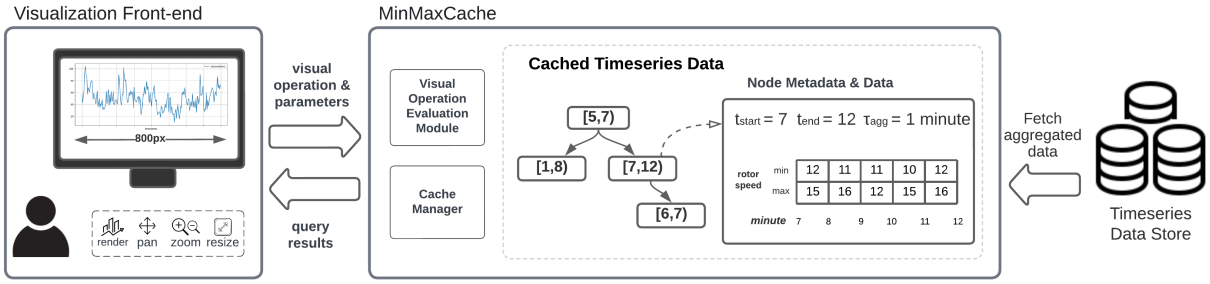


Figure 2: Overview of the in-memory cache design and its use.

Mapping data points to pixels. We assume that the boundaries of the visualization canvas of width w and height h pixels are determined by the min and max timestamps and values of the time series. The canvas refers to the area dedicated to rendering the actual time series line, excluding elements such as axes, labels, and other non-data components. The line chart is generated by mapping data points to discrete pixels (i.e., foreground pixels) and rasterizing the lines between consecutive data points to create a continuous, connected representation of the time series. For the mapping of a data point (t, v) to its corresponding pixel $(p_x(t), p_y(v))$, most visualization clients employ the following geometric transformation:

$$p_x(t) = \lfloor w \cdot \frac{t - t_{start}}{t_{end} - t_{start}} \rfloor, \quad p_y(v) = \lfloor h \cdot \frac{v - v_{min}}{v_{max} - v_{min}} \rfloor \quad (1)$$

For simplicity, we focus on two-color, or binary line charts, which use a color (e.g. black) for foreground pixels and white for background. We also experimented with anti-aliased techniques that render foreground pixels with different hues, finding that the error was smaller than of binary ones, as it is also reported in [15].

A *pixel column* is a one-pixel wide vertical slice of the canvas, spanning its entire height (h pixels), rendering the minimum time interval in the line chart visualization. Foreground pixels are rendered either due to *inner-column lines*, connecting data points mapped to the same pixel column, or due to *inter-column lines*, connecting points across columns. Fig 3a shows in grey the foreground pixels mapped from raw data points in a time series.

Based on the formula above, the range of inner-column foreground pixels depends on the values of the data points within the time interval of the pixel column, irrespective of their timestamps. It spans from the bottom pixel, derived from the minimum data point value, to the top pixel, derived from the maximum value. In contrast, the inter-column foreground pixels are determined by the line segment connecting data points across adjacent columns. This segment may traverse additional pixels beyond the inner-column range, depending on the timestamps and values of these data points.

Visual Operations. We consider an exploration setting where a user visually explores a time series dataset on a $w \times h$ pixel canvas, using interactive operations, such as panning (shifting the visualized time interval), zooming in/out (altering the interval size), showing/hiding a variable, and resizing the canvas dimensions. While operations like similarity queries and pattern extraction can be considered, our focus is on a *locality-based* exploration scenario involving overlapping time series regions.

3.2 Determining Upper Bound for Pixel Errors

In this section, we first define the notion of equidistant groups, which is used as an aggregation method for reducing the number of data points at the scale of the pixel columns; then we employ this method to create groupings with smaller time intervals than a pixel column width and define the notion of fully and partially overlapping intervals. Finer min-max aggregations of data can be cached and reused for subsequent visualizations without querying the database. This method however may introduce errors. We, thus, present two types of errors and calculate an upper bound based on cached data in finer-than-pixel-wide groupings. We show that this bound can be computed based on the already cached data, without resolving to raw data in the database (Theorems 3.3 and 3.4).

Definition 3.1 (Equidistant Time Series Grouping). Given a time series T of length n, an *equidistant* grouping of T in k non-overlapping groups is denoted by $\mathcal{G}^k(T) = (B_1, B_2, \ldots, B_k)$. Each group B_i contains the data points with timestamps in its interval, such that $B_i = \{(t_j, V_j) \mid t_j \in [t_1 + (i-1)\tau, t_1 + i\tau)\}$ for $i = 1, 2, \ldots, k-1$ and $B_k = \{(t_j, V_j) \mid t_j \in [t_1 + (w-1)\tau, t_n]\}$, where t_1 and t_n are the starting and ending times of the time series T, and $\tau = \frac{t_n - t_1}{k}$ is the length of the interval of each group.

Fig. 3a depicts a 4×4 pixels chart. M4[15] defines that an error-free visualization can be obtained by grouping the data in w equidistant groups and mapping the data points corresponding to the first and last timestamp, and to the min and max values, to pixels. In this case, we employ a grouping $\mathcal{G}^4(T)$ since $w = 4$. The resulting visualization is the same with the one produced by retrieving and rendering to pixels all data points in the time series (Fig 3a depicts an error-free visualization produced by raw data; the data points that would be selected by M4 are highlighted in green).

Definition 3.2 (Fully-contained and Partially-contained Intervals). Let $\mathcal{G}^w(T)$ and $\mathcal{G}^k(T)$ be two equidistant groupings of a time series T, with w and k representing the number of groups in each grouping, respectively. We define a group B_i^k in $\mathcal{G}^k(T)$ to be *fully-contained* in a group B_j^w in $\mathcal{G}^w(T)$ if its time interval is completely contained within the time interval of B_j^w. We denote this as $B_i^k \subseteq B_j^w$. A group B_i^k is *partially-contained* in a group B_j^w if their corresponding time intervals intersect. We denote this as $B_i^k \cap B_j^w \neq \emptyset$ and $B_i^k \not\subseteq B_j^w$.

Fig. 3b depicts a secondary grouping $\mathcal{G}^7(T)$ of 7 groups overlapping (either fully or partially) with the $\mathcal{G}^4(T)$ pixel-based grouping.

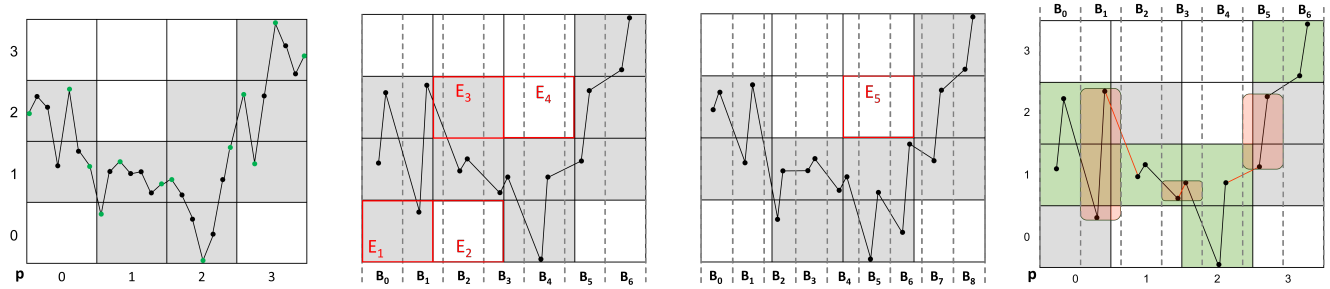


Figure 3: Pixel errors: White rectangles for background pixels, gray for foreground, and red borders for errors. Visualization using: (a) raw data, while green are the points selected by M4; (b) Min Max Aggregation with k=7 groups, having four pixel errors (dashed lines mark grouping intervals); (c) k=9 groups, showing only one error, demonstrating improved accuracy with finer granularity. Additionally, (d) examines potential pixel errors: Green rectangles mark correct foreground pixels from fully-contained groups. Red zones highlight areas of potential inaccuracies in partially-contained groups that may lead to both inner- and inter-column errors. Red lines depict incorrect inter-column lines that can lead to false inter-column pixels.

Min Max Aggregation with Mid-Interval Timestamps. Our method groups data in groupings with smaller intervals than the pixel width, resulting in partially and fully contained intervals in the pixel-wide ones. Also, in contrast to the M4 methods presented in [15], it retrieves and caches for each group only two values, the **min** and **max** value of y, *ignoring the timestamps* of these datapoints, as well as the first and last data points in this group. We arbitrarily assign the middle of the group's interval as a timestamp to these two values. In the case of a *partially contained* group B_i^k within two consecutive pixel columns B_l^w and B_j^w, the selection of a timestamp representative can influence the determination of the pixel column, which the group's min and max y values are mapped to. Selecting the middle of the group's interval ensures that the majority-overlapping pixel column maps the min and max y values of the group. Fig 3b shows our approach with the min and max values for the $\mathcal{G}^7(T)$ grouping, overlapping the pixel columns.

This approach speeds up queries both in relational and time series databases, as aggregate values, such as min and max, are usually precomputed in table statistics and can be fast retrieved. However, to select the specific datapoints (timestamps) that correspond to the min and max values of k groups requires to lookup these values in the underlying time series table[2].

Errors due to Min Max aggregation. Using the above aggregation method, we can utilize the min-max values of the fully-contained groups, to accurately determine at least a subset of the inner-column pixels. Since these groups are completely within a pixel column, their min-max ranges fall within the min-max range of the pixel column, thus contributing to the range of inner-column pixels.

However, in our approach, both *inner-* and *inter-column* pixel errors may occur. Inner-column errors arise when a group is partially-contained in two adjacent pixel columns. Using the mid-interval timestamp for a group's min and max values (instead of their actual timestamps) may lead to either mistakenly omitted or falsely added foreground pixels for the columns overlapping the smallest and biggest part of the group, respectively. In the first case, the min-max range for the pixel column might be smaller than its actual range, while in the second, errors may arise when the partially contained group's min-max range exceeds that column's actual range.

On the other hand, inter-column errors result from our aggregation method's omission of the first and last data points in each pixel column, leading to incorrect inter-column line segments. These errors occur when connecting the last min or max values from groups in one column to the first of such values in the adjacent column. For example, in Figure 3b, errors E_1 and E_2 could be avoided with correct inter-column line rasterization, while E_3 and E_4 also occur due to difficulties in determining the accurate min-max range within the first two pixel columns because of the partially-contained group.

Based on the types of errors, it's evident that *increasing the number of groups can generally reduce the errors*. Decreasing the group size relative to the pixel columns results in increasing the fully contained groups and thus mapping more min max points to a pixel column. Although the maximum number of partially contained groups remains constant—at most one between two adjacent pixel columns— still the area that each partially contained group overlaps with the pixel columns is reduced; therefore contributing to less uncertainty. This result is illustrated in Figure 3c, which shows a decrease in pixel errors from 4 to 1 as the number of groups (k) increases from 7 to 9 compared to Figure 3b.

Error bounds. The following two theorems provide the upper bound for inner and inter-column errors. The first theorem calculates the inner-column error bound by evaluating the differences between the ranges of partially and fully contained groups within a pixel column. The second theorem assesses potential inter-column errors, focusing on inaccuracies at intersections between adjacent columns. It measures the divergence of our approximated from the actual data lines, accounting for both falsely added and mistakenly omitted foreground pixels at these intersections.

Theorem 3.3 (Inner-column errors). *Assume a two-color line chart of a time series variable y generated using the min and max values for y on a grouping $\mathcal{G}^k(T)$. The potential inner-column pixel errors E^i_{inner} in a pixel column p_i is the set difference of the pixel*

[2]In our experiments, we found that, both in PostgreSQL and InfluxDB, fetching the timestamps of the min and max values typically resulted in doubling the query duration.

ranges P of the left and right partially overlapping groups l, r ($B_l^k \cap B_i^w \neq \emptyset$, $B_r^k \cap B_i^w \neq \emptyset$, and $B_l^k \not\subseteq B_i^w$, $B_r^k \not\subseteq B_i^w$) and the pixel range of the fully-contained groups in p_i, respectively. Given the following pixel ranges:

$$P_l = [p_y(B_l^{min}), p_y(B_l^{max})], \quad P_r = [p_y(B_r^{min}), p_y(B_r^{max})],$$

$$P_i = \bigcup_{B_j^k \in \mathcal{G}^k(T) | B_j^k \subseteq B_i^w} [p_y(B_j^{min}), p_y(B_j^{max})]$$

The potential inner-column pixel errors are given by

$$E_{inner}^i = (P_l \cup P_r) \setminus P_i$$

Proof. Consider a pixel column p_i in a two-color line chart of the time series variable y. Let P_i be the set of foreground pixels derived from min and max y values of fully-contained groups within p_i, and P_l and P_r the sets of potential foreground pixels contributed by the left and right partially-overlapping groups l and r.

Using the known min and max y values for l and r, these sets correspond to the following pixel ranges determined using Eq. 1:

$$P_l = [p_y(B_l^{min}), p_y(B_l^{max})], \quad P_r = [p_y(B_r^{min}), p_y(B_r^{max})].$$

The foreground pixels in p_i must cover at least P_i. However, data points within the min and max values of partially-overlapping groups may potentially contribute to p_i, with an upper limit of $(P_l \cup P_r) \setminus P_i$. To account for timestamp inaccuracies (using the midpoint of the interval), we consider the worst-case scenario, resulting in a maximum set of additional potential foreground pixels, which is equivalent to $(P_l \cup P_r) \setminus P_i$. Thus, the maximum set of potential inner-column pixel errors E_{inner}^i corresponds to $(P_l \cup P_r) \setminus P_i$. □

To understand missing inner-column pixels, we examine the partially-contained groups intersecting the pixel column boundaries. Figure 3d illustrates how we determine potential errors, using the same raw time series data depicted in Figure 3a, and the same min-max aggregation on $\mathcal{G}^7(T)$ as shown in Figure 3b.

In the figure, gray pixels are foreground pixels resulting from intersecting the line, while green pixels are correctly visualized inner-column pixels from fully-contained groups. For example, pixels $p_{1,0}$ and $p_{2,0}$ are correct foreground pixels, as the first group of the grouping is fully-contained within the first pixel column, and its min-max values are guaranteed to be included in the min-max value of the overall pixel column. These are inner-column pixels.

However, partially-contained groups, such as B_1 group, present challenges as the exact mapping of min-max values to the two adjacent pixel columns is uncertain. If the actual data point with the min or max values lie in the second pixel column and its range exceeds the min-max range of all fully contained groups in the pixel column, then exceeding inner-column pixels will not be rendered (missing inner-column pixels). For instance, considering Figure 3a, $p_{0,1}$ is an inner-column pixel that is missing from the line chart generated using $\mathcal{G}^7(T)$ (Fig. 3b). Simultaneously, because we approximate the timestamp with the middle of each group, we have a false inner-column pixel ($p_{0,0}$) that is rendered as a foreground by incorrectly including within it the min value of the partially-contained group.

To illustrate potential inner-column pixel errors, red zones are drawn over Figure 3d. These areas cover the partially-contained groups and are vertically delimited by their min-max ranges. Pixels overlapped by these zones and not green (i.e., true foreground pixels derived from the fully-contained groups) are considered as potential inner-column pixel errors. Thus, we can determine that pixels $p_{0,0}$, $p_{0,1}$, $p_{2,1}$, $p_{2,2}$, $p_{2,3}$, and $p_{1,3}$ are potential inner-column pixel errors.

Theorem 3.4 (Inter-column errors). *For two adjacent pixel columns p_i and p_j in a two-color line chart generated using the min-max values of a variable y on a grouping $\mathcal{G}^k(T)$, the potential inter-column pixel errors for column p_i because of a missing or false inter-column line between p_i and p_j are given by:*

$$E_{inter}^{i,j} = (F_{i,j} \cup M_{i,j}) \setminus P_i$$

where $F_{i,j}$ denotes the pixels in column p_i, resulting from rasterizing the line segment between the min or max y values of $\mathcal{G}^k(T)$ before and after the $p_i - p_j$ intersection, $M_{i,j}$ the maximum set of pixels potentially affected by the missing correct inter-column line, and P_i the set of inner-column foreground pixels that can be accurately determined by the fully-contained groups of $\mathcal{G}^k(T)$ in column p_i.

Proof. In a two-color line chart using min-max y values from a grouping $\mathcal{G}^k(T)$, the correct inter-column line between adjacent pixel columns p_i and p_j cannot be determined. Instead, a line connecting the min or max y values across their intersection is rendered, which can introduce both missing and false foreground pixels in p_i.

Define $F_{i,j}$ as the set of pixels in p_i affected by this approximation, determined by rasterizing this false line and identifying intersecting pixels in p_i. Excluding accurately determined inner-column foreground pixels P_i from fully-contained groups, the potential false foreground pixels due to the approximation are $F_{i,j} \setminus P_i$.

Let $M_{i,j}$ denote the maximum set of potentially missing foreground pixels in column p_i due to the absence of the correct inter-column line. This calculation varies based on two scenarios: (i) With a single partially-contained group (B_{ij}) at the intersection of p_i and p_j, missing pixels are within B_{ij}'s vertical range, defined as $M_{i,j} = [p_y(B_{ij}^{min}), p_y(B_{ij}^{max})]$. (ii) Without a partially-overlapped group, but with fully-contained groups adjacent across the intersection (B_i in p_i and B_j in p_j), the missing line may start from any point in B_i and extend towards B_j. The potential missing pixels are bounded within the largest possible ranges defined by the min and max values of adjacent groups, resulting in $M_{i,j} = [p_y(B_i^{min}), p_y(B_j^{max})]$ or $M_{i,j} = [p_y(B_i^{max}), p_y(B_j^{min})]$. Excluding accurately determined inner-column foreground pixels P_i, the maximum potential missing foreground pixels in p_i are $M_{i,j} \setminus P_i$.

Combining both potential false and missing pixels, the maximum inter-column pixel errors in column p_i due to the inter-column line approximation at $p_i - p_j$ are represented by $(F_{i,j} \cup M_{i,j}) \setminus P_i$. □

The theorem aims to measure the maximum number of pixel errors that arise between two adjacent pixel columns. In Figure 3d, the inter-column line segment between pixel columns 0 and 1 (depicted by the red line segment) designates pixel $p_{2,1}$ as a foreground pixel (shown in gray). However, this pixel is wrongly set as a foreground pixel ($E3$ in Figure 3b). since the inter-column line does not match the one in the raw data visualization (Fig 3a).

Conversely, not rendering the correct inter-column line may lead to missing pixels. For example, $p_{2,2}$ is not set as a foreground pixel as it is not crossed by an inner or inter-column line, although the

correct inter-column line does pass from this pixel. This potential missing pixel corresponds to an actual error (E_4 in Fig. 3b).

Next, we combine Theorems 3.3 and 3.4 to define the upper error bound as *the proportion of total pixels in the visualization canvas that could be incorrect*. Note that, we do not use SSIM (Structural Similarity Index Measure) [37], as done in [15, 36], for determining the upper error bound. SSIM measures image similarity by evaluating changes in structural information, brightness, and contrast, focusing on spatial pixel interdependencies to approximate human visual perception. However, it requires a reference image (error-free) to compute similarity, which we lack during query evaluation. Instead, our theorems provide the maximum potential pixel errors, allowing us to quantify the inaccuracy introduced by the aggregation method without comparing to raw data visualization. This method, unlike SSIM, estimates absolute pixel errors without explicitly considering structural information and the human perception model. However, in our experiments, we evaluate SSIM and show that our calculated error bound is proportional to this metric.

Definition 3.5 (Upper Error Bound). Given a two-color line chart visualization of a time series variable y with a width w and height h pixels, let E^i be the set of inner- and inter-column pixel errors in the i^{th} pixel column, i.e., $E^i = E^i_{inner} \cup E^{i-1,i}_{inter} \cup E^{i,i+1}_{inter}$.

The use of sets and their unions in this calculation is to ensure each pixel is counted only once, avoiding redundancy in cases where a pixel could be part of both inner- and inter-column errors. Then, the upper error bound is the ratio of the total number of errors in all pixels columns to the number of pixels in the visualization:

$$\epsilon = \frac{\sum_{i=1}^{w} |E^i|}{w \times h} \tag{2}$$

Problem Statement. Given a multi-variate time series T of length n, a visualization canvas of width w and height h, a sequence of visual interactions over the line chart visualization of T with error constraints $\leq \epsilon$ with respect to the visualization accuracy, our problem is *an approximate visualization problem with pixel error-bound guarantees*, which aims at providing interactive response time at each user operation ($\sim$ 500ms [23]). This problem encapsulates the challenge of balancing the interactivity and the accuracy of the visualization, particularly in successive user interactions.

4 VISUALIZATION-AWARE CACHING

This section presents the proposed cache, query evaluation over it, as well as the eviction and prefetching techniques employed.

4.1 MinMaxCache Design

Multi-Time Series Caching. Figure 2 depicts the structure of the cache. It uses an interval tree [9] for indexing the data. A node in the tree represents a grouping $G^k(T)$, and stores its start and end timestamps, aggregated metadata about its intervals as well as pointers to its left and right subtrees. This tree allows to efficiently find overlapping cached time intervals with the query interval. The cache accommodates multiple variables in a time series by utilizing separate interval trees for each variable. A hash table maps each unique combination of a time series key $\mathcal{T}_{id}$ and a specific variable y to its corresponding interval tree. Employing separate interval trees allows for more fine-grained control on how the data is fetched, aggregated and cached, adjusting the unique patterns of variables to different caching and aggregation granularities as well as to different error bounds. Consequently, this method enhances MinMaxCache's efficiency, as it aims to fetch only the data necessary for the user's current focus.

Cache Metadata. A node in the tree corresponds to a grouping $\mathcal{G}^k(T')$ of the subseries $T' \subseteq T$, where $T' = \{(t_j, V_j) \mid t_j \in [t^{\mathcal{G}}_{start}, t^{\mathcal{G}}_{end}]\}$, and $k = \lceil \frac{t_{\text{end}} - t_{\text{start}}}{\tau_{\text{agg}}} \rceil$. Each node $\mathcal{G}$ stores:

- $\mathcal{T}_{id}$: The unique identifier of the time series interval tree to which the node belongs.
- y: The specific variable of the time series for which the node stores aggregated min/max values, with $y \in Y$, where Y is the set of all variables of the multivariate time series.
- $t^{\mathcal{G}}_{start}, t^{\mathcal{G}}_{end}$: The start and end timestamps of the grouping.
- τ_{agg}: The aggregation interval, i.e., the length of each k time interval in the grouping.
- $\mathcal{V}_{min}$ and $\mathcal{V}_{max}$: An array of k size with the min and max values of the variable y for each interval in the grouping.

To calculate the i^{th} time interval in the grouping $\mathcal{G}$, we use:

$$(t^i_{start}, t^i_{end}) = (t^{\mathcal{G}}_{start} + i \cdot \tau_{agg}, \min(t^{\mathcal{G}}_{end}, t^{\mathcal{G}}_{start} + (i+1) \cdot \tau_{agg})$$

4.2 Query Evaluation over MinMaxCache

In MinMaxCache, a query is the result of a user operation on the visualized time series T, from which the user selects a subset of numeric variables $Y_Q \subseteq Y$, the query time interval $I_Q = [t^Q_{\text{start}}, t^Q_{\text{end}})$ and the width w and height h in pixels of the visualization canvas.

Figure 4 shows an example of a query Q to generate a visualization with $w = 3$ pixels. The process involves the following steps: the evaluation of the query on the cached data (step 1); the evaluation of the error bound based on the cached data and fetching of missing data (step 2); the final evaluation of the error and visualization of the results (step 3). Note that these steps are applied to each interval tree of a variable in Y_Q. However, data access is optimized by issuing a single query to fetch missing data for all variables Y_Q.

Evaluate the query on the cache. In Step 1, the interval trees that match the requested time series and variables in Q are identified via their unique identifiers, $\mathcal{T}id$ and $y \in Y_Q$. For each variable y in Y_Q, its interval tree is either accessed or constructed (the first time is visualized). The groupings that overlap with the query interval, I_Q, are identified in the interval trees [9]. Specifically, only groupings with an aggregation interval $\tau_{agg} \leq \tau_p/2$ are considered, where τ_p, representing the pixel column interval size, is defined by the query interval as $\tau_p = (t^Q_{\text{end}} - t^Q_{\text{start}})/w$. This requirement arises from the necessity to have at least one fully contained group in every pixel column. Otherwise, it is not possible to identify a subset of correct inner-column pixels; and thus, the error bound calculated based on Theorems 3.3 and 3.4 could be excessively high.

The process, then, identifies for each variable the parts of the query interval, for which no cached groupings exist and access to the data is needed. In Figure 4, $G1$, and $G2$ cover only the blue parts (first 2 pixel columns) of the query interval.

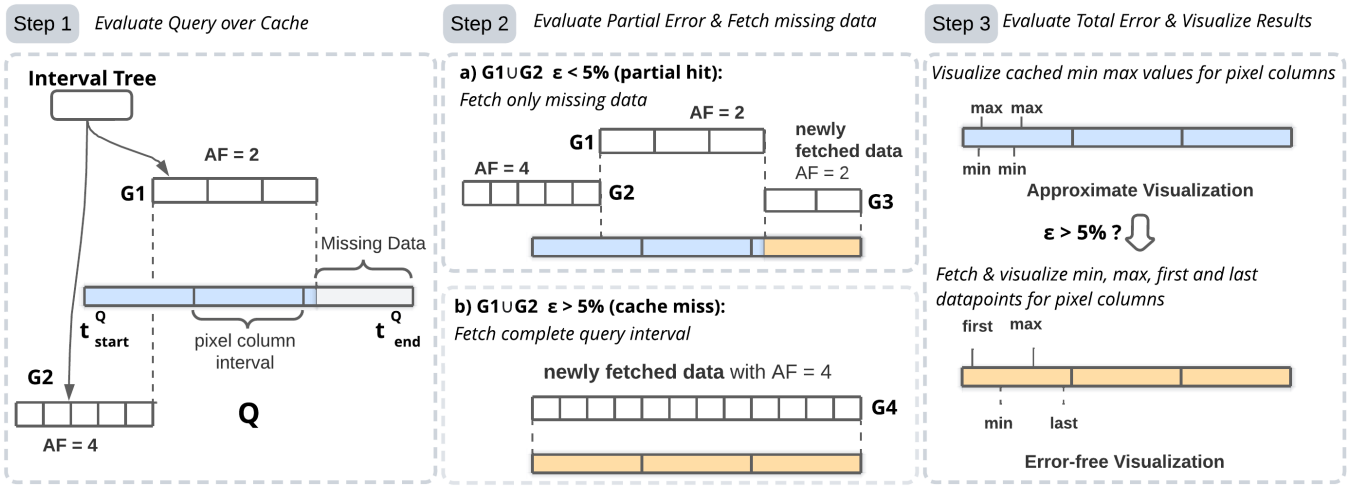


Figure 4: Evaluation of a query Q for a visualization with $w = 3$ pixels. Overlapped groupings $G1$ and $G2$ are retrieved from the interval tree. Based on partial error, it can be a partial hit (case a), fetching only missing data, or a cache miss (case b), triggering a query for the entire interval. In Step 3, total error ϵ is evaluated using cached and fetched data. If error still exceeds the bound, we fetch min, max, first, and last data points for each pixel column, ensuring an error-free visualization.

Evaluate error and fetch missing data. Step 2 concerns the evaluation of the error bound using the Theorems 3.3 and 3.4. We distinguish three cases: i) The cache holds data for the entire query interval across all variables. We compute a separate error bound for each variable. If errors are within the constraints of each variable, we compute the min-max values of each pixel column based on the contained groupings and visualize the results. We consider this case as *complete cache hit*; for space reason, this type of query is not shown in the Figure. ii) The cache misses data for parts of the query interval of a variable. Again, we compute the error bound based on the groupings overlapping the query interval. If it is within the expected limit (e.g., $\leq 5\%$), we only fetch the missing part of the query from the datastore. This is a *partial cache hit* (Figure 4 Step 2a). Fetched data is aggregated into new groupings in their corresponding interval trees. In the example, the new grouping $G3$ is cached into the variable's tree and used for computing the min-max values of pixel column 3. iii) Finally, if the partial or complete error bounds exceed constraints, we fetch data for the entire query interval, employing a new aggregation interval $\tau'_{agg} < \tau_{agg}$, which reduces the error (Figure 4 Step 2b) . This case is a *cache miss*. The new aggregation interval is determined by the Aggregation Factor.

Definition 4.1 (Aggregation Factor). The Aggregation Factor AF, is the ratio of the pixel column interval τ_p to the aggregation interval of the grouping, τ_{agg}, i.e., $AF = \frac{\tau_p}{\tau_{agg}}$. AF ≥ 2, since $\tau_{agg} \leq \tau_p/2$.

This metric quantifies the granularity of a grouping interval to the pixel column interval. Determining a granularity for a new grouping that would satisfy the error constraint, is particularly challenging. It depends not only on the query interval and the resolution of the visualization, but also on data characteristics, which in streaming data scenarios cannot be usually determined upfront. Also, the chosen factor must facilitate cached data reuse in future queries, which are often not known beforehand, considering the dynamic nature of exploratory workloads.

To address these complexities, MinMaxCache employs an adaptive approach, considering the aggregation factor of cached groupings overlapping the query. When multiple groupings overlap, the aggregation factor of the grouping covering the largest portion of the query interval is used to issue a query to the database with a doubled aggregation factor. This method offers an *adaptive* way of extending the cache with finer-granularity groupings for the area the user explores. It enhances the likelihood of fetched results satisfying the error bound, as well as the probability that these results will comply with the error bound for future user operations around the area (results are provided in experiments). For example, grouping $G1$ covers a larger part of Q compared to $G2$, having $AF_1 = 2$, i.e., each pixel column interval is twice the length of each group interval in $G1$. In case (a), with a partial hit and error $\leq 5\%$, we fetch the missing data with $G1$'s AF; otherwise (case b), we fetch fine-grained data by doubling the value of $G1$ ($G4$ with AF=4).

When the cache lacks overlapping groupings for a new query, an initial aggregation factor is set. Our experimental evaluation indicates that while MinMaxCache can adaptively adjust the AF value, leading to consistent performance, an initial value of 4 tends to yield optimal performance in most cases (Sec. 5.2).

This adaptive approach, including the computation of the aggregation factor, is performed per variable. For example, a variable might not violate the error bound and therefore does not require the doubling of the aggregation factor. However, a single database query is issued to minimize round trips and utilize database optimizations. This unified query efficiently collects required data for all variables, despite varying aggregation intervals.

Evaluate total error and visualize the results. After the query is executed and the new groupings are cached, Step 3 concerns the evaluation of the total error bound considering the new data fetched. If the reevaluation still exceeds the error bound for some variables, a final M4 query guarantees 100% accurate visualization. Although

this can result in increased total query time, our experiments indicate that this occurs in less than 5% of the queries. Importantly, even with this two-step access to the data store, users can visualize intermediate results before examining the results from the second query, which are guaranteed to be 100% accurate.

To avoid the unnecessary computational cost of aggregating data with an aggregation interval close to the sampling interval τ_s, we fetch and cache raw data when $\frac{\tau_{agg}}{\tau_s} < 6$. Yet, in our experiments, such instances were very rare, only arising during highly zoomed-in exploration, where the pixel column interval approached τ_s.

Step 3's final stage involves mapping the cached data of each variable in Y_Q to pixel columns and visualizing the results. The process iterates over the grouping intervals in each pixel column for every variable, returning four key points: the minimum, maximum, first and last values (the latter computed from the timestamps of the first and last groupings within each pixel column).

The *cache size* for a single variable y depends on the number of cached groupings and the groups within each. Each group stores two min-max values for y, with the number of groups k determined by the Aggregation Factor used during data retrieval, aligning with the visualization's resolution ($O(AF \cdot w)$). The number of cached groupings is primarily driven by the query sequence. The worst-case scenario arises when all queries result in cache misses, necessitating new groupings with a doubled Aggregation Factor. Overall, the cache size is bounded by the raw data ($O(n)$), but fetching raw data is very rare, mainly occurring during extensive zooming.

Complexity of Query Evaluation. Query evaluation complexity depends on database access. For cache hits, it involves interval tree searches ($O(\log g + o)$, with g being the total groupings and o the overlapping ones) and processing groups for error bound calculation ($O(w)$ to $O(n_Q)$, where n_Q is the number of data points in the query interval, with the worst case occurring when the cache stores non-aggregated data). While these operations are performed per variable, the number of simultaneously visualized variables can be considered constant. The overall worst-case complexity simplifies to $O(n_Q)$, considering w is limited by display resolutions and n_Q is usually larger than g. In partial cache hits or misses, complexity is driven by database fetches, scaling with the time series size ($O(n)$), which also applies to M4 queries.

4.3 Cache Eviction Policy

MinMaxCache employs an eviction policy to manage the memory footprint. The policy targets locality-based exploration scenarios, evicting nodes less likely to be used in subsequent user operations that involve overlapping regions. It calculates the distance between the centers of a cached grouping node and the interval of the last visualized query, assigning the value of 0 if the grouping is entirely within the query interval. The replacement policy operates by evicting the node with the greatest distance when the cache is full.

4.4 Prefetching

There exist several prefetching approaches that model user behavior and predict future operations and fetch data that might be needed in the future [2, 5, 16]. MinMaxCache employs a simple prefetching strategy, prefetching data to the left and right of the most recent query. This strategy accelerates pan and zoom out operations; the latter commonly use a zoom factor of 2 and usually result in additional data fetching. Then, we prefetch data, which are not already cached with as much detail (i.e., aggregation granularity) as in the last query. The aggregation granularity used for prefetching aligns with the granularity of the previous query if the fetched data met the error bound. Otherwise, we fetch data with the new aggregation factor, adhering to our adaptive caching approach.

Prefetching can notably improve performance by reducing the need for future data fetches. However, it could potentially increase the latency of both the user's query and the final visualization, since resources are allocated to fetch data that may not be immediately necessary. To ensure fast and interactive querying, we perform prefetching during user idle times, i.e., while the user is examining the results of their prior operation. This method minimizes the impact of prefetching on system responsiveness and allows uninterrupted user interaction (see evaluation details in Section 5).

5 EXPERIMENTAL ANALYSIS

The goal of our evaluation is to assess MinMaxCache's performance in terms of response time and accuracy of the resulting visualization. We evaluate MinMaxCache with respect to its various parameters and against several competitors using both real and synthetic data.

5.1 Experimental Setup

Datasets. Our evaluation uses the datasets in Table 1: (*i*) *Real datasets*: We used three datasets: *MNF* which contains data from an electrical power sensor of a semiconductor manufacturing machine (sensor MF03)[14], *SOCC* which contains measurements from a speed sensor of a soccer ball [30], and *INTL* which includes readings collected from sensors at the Intel Berkeley Research Lab [19]. Given their smaller size, we expanded each by repeating the data 50 times. (*ii*) *Synthetic datasets*: We generated 11 synthetic datasets using random walks, starting at 1 million and scaling up to 1 billion datapoints. The sampling interval was adjusted with each dataset to double its size from the previous one, all representing a 4-year span. These datasets are collectively referred to as *SNT1M-1B*.

Table 1: Dataset basic characteristics

Name	Sampling Interval (s)	# Fields	# Datapoints
MNF	0.01	16	20M
INTL	30	4	115M
SOCC	0.5	11	350M
SNT1M-1B	0.125-120	5	1M-1B

Evaluation Metrics. The performance and quality of visualizations are evaluated using the following measures: *(a) Query Response Time:* total time taken by *MinMaxCache* to evaluate the user operation not including the rendering of the visualization; *(b) Visualization Quality*: similar to [15], we use the *normalized structural similarity index* (SSIM) [37] to measure the similarity between visualizations generated by *MinMaxCache* and those using raw data.

Competitors. We compare our approach with: (1) M4 [15], (2) OM3 [36], and (3) RawCache. The latter is a variation of our in-memory cache which fetches and caches non-aggregated data points, similar

to TSCache, [22] (code was not made available for evaluation). RawCache serves as a baseline to demonstrate the limitations of a visualization-agnostic caching approach.

Exploration Scenario. In our evaluation, we simulated a user exploration scenario involving sequences of 50 pan and zoom in/out operations. Each query interval was either shifted (i.e., pan operation) by 10-50% or zoomed in/out by a zoom factor of 2 relative to the preceding one. The next operation was randomly determined with 35% probability for pan left, 15% for pan right, 20% for zoom in, and 30% for zoom out. By default, our exploration scenario assumes that the user visualizes a single variable of the time series data, while a multi-variate scenario is considered in a separate experiment. The first query was placed at the end of the data to mirror a typical exploration scenario where the user begins from the most recent data and navigates towards older data or zooms out for a broader perspective. We define query selectivity as the percentage of data points selected by the query relative to the total number of data points, assuming a value of 10% for the first query. Finally, unless stated otherwise, we use a default error bound (ϵ) of 5%.

Implementation Details. MinMaxCache is implemented on JVM 11. As a database, we employed PostgreSQL(v.15.2) which was also used for M4 [15], and InfluxDB(v.2.6.1), a time series database also considered in [36]. For visualization rendering, the Cairo(v1.17.8)[3] library was used. For our evaluation, the database was hosted in an Ubuntu 20.04.06 server with an AMD Ryzen Threadripper 3970X 32-Core Processor and 252GB RAM and 10TB SSD, and MinMaxCache run on a MacBook Air 2020 laptop with an M1 processor and 16GB RAM and 512GB SSD. Both devices were on the same network.

5.2 MinMaxCache Parameters

This section evaluates MinMaxCache's performance regarding the error bound constraint, initial aggregation factor, and prefetching. We also assess MinMaxCache's visualization accuracy, focusing on results using InfluxDB, with similar outcomes for PostgreSQL.

Performance vs. Error Bound. In this experiment, we analyze how varying the error bound ϵ affects MinMaxCache's performance. Focusing on MNF on InfluxDB, Figure 5 shows execution times for different error bounds. Higher ϵ generally shortens query execution times as it allows fetching coarser data with lower latency while still meeting the error constraint. It also enables more cached data reuse and reduces processing and transfer times. Conversely, a tighter 1% error bound increases evaluation times, often requiring fetching aggregated data with a doubled *AF* or even a second database trip to issue an M4 query, unlike with 5% or 10% bounds.

Varying the Initial Aggregation Factor. We conducted experiments assessing how the initial Aggregation Factor (*AF*) affects MinMaxCache's performance, testing initial *AF* values of 2, 4, and 8. Recall that *AF* is the ratio of pixel column interval to aggregation interval and determines data granularity relative to pixel column interval length. Figure 6 presents the MNF dataset results in InfluxDB, as other datasets show similar behavior. An initial *AF* of 4 slightly outperforms 2 and 8 in cumulative time for the workload. A lower initial *AF* fetches coarser data, potentially failing to meet the error bound, requiring a doubled aggregation factor for fetching the entire query interval. Conversely, an *AF* of 8 may fetch excessively fine-grained data initially, increasing database query latency.

Examining query 0 with a cold cache highlights *AF*'s impact on latency. Initially, a higher aggregation factor raises query times due to more processing and database data transfer. However, this trend does not generally hold in subsequent queries, since cached data can be reused, especially when it's finer-grained, reducing the interval that needs fetching from the database. For example, in query 2, an initial *AF* of 8 requires less time as finer-grained cached data can be used for the non-cached interval. Conversely, *AF* values of 2 and 4 necessitate retrieving the entire query interval with a doubled *AF* due to the error bound data violation.

However, when considering the cumulative times for the entire workload for all aggregation factors, we see that its effect is not so significant, since *MinMaxCache's ability to adaptively determine the appropriate granularity yields consistent performance across the majority of queries, regardless of the initial aggregation factor.*

Prefetching Strategy. In this experiment, we evaluated the impact of prefetching on performance. Prefetching fetches data to the left and right of the canvas during user idle time and is particularly beneficial for queries in previously unexplored areas. We vary the prefetch ratio from 0% (no prefetching) to 75%. Figure 7 shows the cumulative time for increasing prefetch ratios. We present the results for InfluxDB using the INTL dataset for brevity, though similar behavior was observed in the other settings. As anticipated, larger prefetch ratios result in smaller evaluation times. However, values beyond 50% do not yield further time improvements, as each query is either shifted (i.e., pan operation) by 10-50% or zoomed in/out by a factor of 2 relative to the preceding one. Should a scenario permit zooming out by more than a factor of 2, larger prefetch values might prove advantageous. In our experiments, *prefetching, on average, improves query performance by about 30%.*

Visualization Quality. Figure 8 shows a box plot of the SSIM values of visualizations generated by MinMaxCache compared to those produced using raw data. We present MinMaxCache's results for all 50 consecutive queries at error bounds of 10%, 5%, and 1%. SSIM increases as the error bound becomes tighter, supporting our case for the effectiveness of MinMaxCache in adhering to error constraints. Furthermore, *by achieving high SSIM values with smaller error bounds, MinMaxCache demonstrates its ability to strike a balance between data accuracy and aggregation efficiency.*

Specifically, for SOCC and INTL, all queries exhibit SSIM greater than 0.99, indicating visualization results nearly identical to precise visualizations. For MNF, SSIM remains high, and any discrepancy with the other datasets can be attributed to the frequent spikes and rapid value changes in this dataset. Considering Theorems 3.3 and 3.4, such variable changes, especially in partially-contained groups or between adjacent columns, may lead to more inner- and inter-column pixel errors. However, as the error bound tightens, SSIM increases, closely approximating an error-free visualization even for MNF's highly variable data.

Figure 9 shows the error bound ϵ value (orange dashed line) calculated for each query in the exploration scenario with a 5%

[3]https://www.cairographics.org/

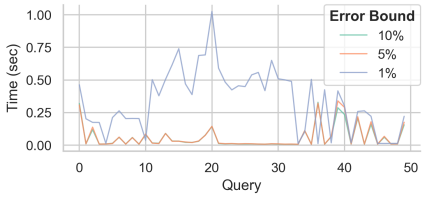


Figure 5: Evaluation Time per query vs. error bound (MNF)

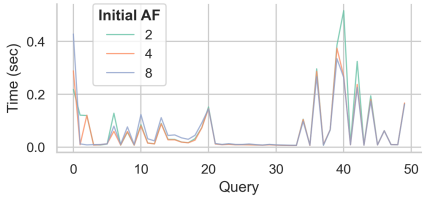


Figure 6: Varying the initial Aggregation Factor (*AF*) for MNF

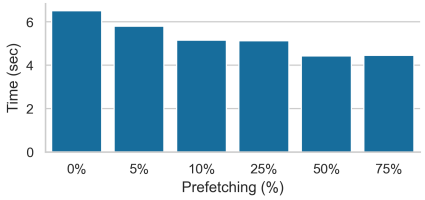


Figure 7: Effect of prefetching on query response time (INTL)

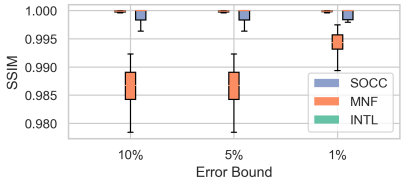


Figure 8: SSIM at different error bounds

Figure 9: Max vs. Actual Error (MNF)

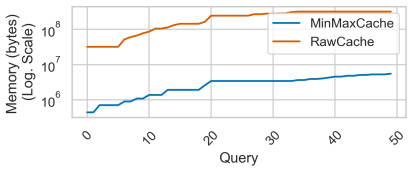


Figure 10: Memory Consumption (MNF)

upper error bound. This value is compared to the actual error, determined by comparing each visualization's pixels with those of the visualization produced using raw data. As depicted, *the actual error consistently falls below the error bound calculated*, demonstrating our method's accuracy in evaluating maximum error.

Memory Footprint. In Figure 10, we evaluate MinMaxCache's memory usage against RawCache for the MNF dataset (with other datasets performing similarly). *MinMaxCache exhibited significantly lower memory usage, being two orders of magnitude more efficient than RawCache.* This arises from our adaptive caching strategy, which fetches data proportional to visualization width using the Aggregation Factor, instead of relying on the total size of the time series as RawCache does. The memory growth is logarithmic, benefiting from effective eviction policies. This pattern indicates that MinMaxCache can handle increasing cached data volumes efficiently, without excessive memory overhead.

5.3 Performance in Exploration Scenarios

Next, we compare MinMaxCache against competitors, with an initial aggregation factor of 4 and prefetching 50% of the data enabled.

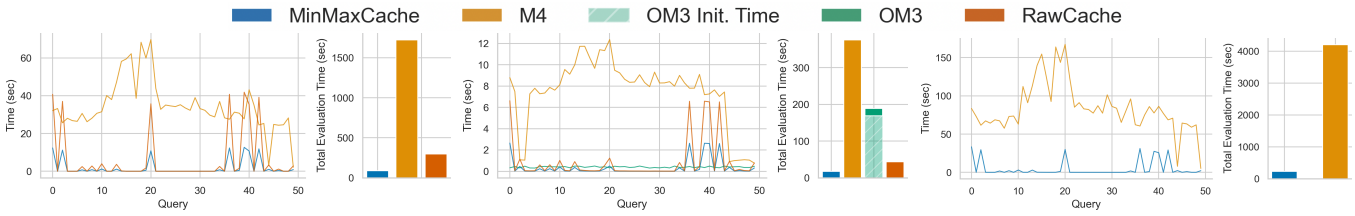
Overall Comparison. Figure 11 shows execution time per query and cumulative time across all three real datasets during the exploration scenario on PostgreSQL. OM3 was only tested on PostgreSQL as there was no open-source InfluxDB implementation available[4]. The results highlight that *MinMaxCache consistently outperforms all competitors in query execution time.* Compared to M4, MinMaxCache performs notably better. M4 queries the database in every user action and, despite returning at most $4w$ tuples, requires aggregating all data within each w interval. MinMaxCache was generally $6-10\times$ faster, with a more pronounced difference in PostgreSQL compared to InfluxDB. This difference was expected, as InfluxDB is optimized for handling large time series data, while PostgreSQL, a generic database, requires complex self-join operations for M4 queries to fetch corresponding timestamps for min, max, first, and last values.

Compared to RawCache, MinMaxCache performs significantly better. RawCache experiences increased latency as it fetches raw data, unlike MinMaxCache's use of aggregated data. RawCache's system failure while processing the SOCC dataset highlights its difficulties in managing large data volumes, while MinMaxCache efficiently caches data provides rapid responses.

OM3 faced heap space limitations indexing datasets exceeding 20 million data points, thus, comparison was restricted to the MNF dataset. As OM3 employs progressive visualization, we report the time needed for OM3 to achieve a SSIM of 1. MinMaxCache displayed promising results compared to OM3, with most queries being faster, except for the initial query from a cold cache and three queries exploring areas with minimal cached data. A key drawback of OM3 is its preprocessing time, highlighted separately in the plot, where constructing the multi-level representation of the time series introduces significant initial delays. Unlike OM3, *MinMaxCache requires no preprocessing, enabling users to immediately start visualizing time series data* and improving overall performance.

Varying the error bound vs. Competitors. Figure 16 displays MinMaxCache's cumulative time against competitors for 10%, 5%, and 1% error bounds for MNF on PostgreSQL. Even at 1% error bound, MinMaxCache significantly outperforms both M4 and RawCache. The same behavior was consistently observed across all datasets and in InfluxDB. Compared to OM3, MinMaxCache exhibits shorter cumulative times in all error bounds, with performance at 1% similar to that of OM3 (19.69 vs. 20 seconds for OM3). However, considering OM3's lengthy initialization time, our approach offers clear advantages. This consistent outperformance across varying error bounds demonstrates that *even in near error-free visualization requirements, (~ 1%) our methods achieves a notable improvement* in the performance of the entire exploration sequence.

[4]https://github.com/Ideas-Laboratory/om3/tree/main



(a) INTL (b) MNF (c) SOCC

Figure 11: MinMaxCache vs. Competitors on PostgreSQL

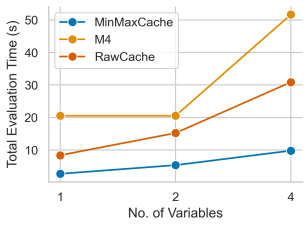


Figure 12: Varying number of variables (MNF)

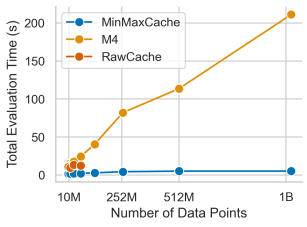


Figure 13: Varying Time Series Length (SNT1M-1B)

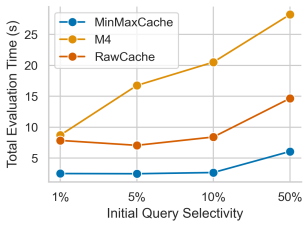


Figure 14: Varying initial query selectivity (MNF)

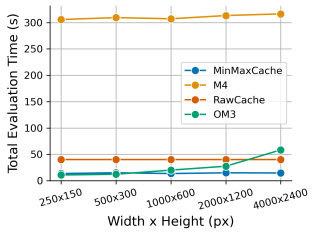


Figure 15: Varying canvas size (MNF)

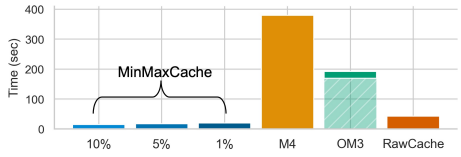


Figure 16: Varying Error Bound vs. Competitors (MNF)

Handling Multi-Variate Exploration Scenarios. Next, we consider the scenario where a user visualizes multiple variables on a multi-series (multiple axes or stacked) line chart, and each operation adjusts the visualization canvas for all variables. Figure 12 presents the total time for evaluating all queries in the default exploration scenario, with respect to an increasing number of variables. We report on the MNF dataset in InfluxDB, with other datasets showing similar behavior. Unfortunately, we were unable to run this experiment for OM3, due to previously mentioned reasons.

As the number of variables increases, query execution time rises. Notably, this increase is more pronounced for M4 and RawCache, whereas less significant for MinMaxCache. Although the time to fetch the aggregated data for multiple variables from the data store is higher, the effective caching per variable mitigates this issue, enabling better management of the data that needs to be fetched from the database and ultimately reducing the overall latency of database access.

Performance vs. Time Series Length. This experiment evaluated MinMaxCache's performance with increasing time series length n, utilizing the SNT1M-1B datasets. Figure 13 displays these results for InfluxDB, revealing that *MinMaxCache scales logarithmically with the time series length*. In contrast, M4 scales linearly due to increasing numbers of datapoints it aggregates per query. However, MinMaxCache effectively reuses cached data, thereby reducing the number of datapoints the database query needs to process and fetch. For RawCache, we could run the experiment up to the size of 64 million data points (it fails for larger datasets).

Varying the initial query selectivity. In this experiment, we evaluate the impact of initial query selectivity—the percentage of data selected, maintaining the same sequence of operations. Figure 14 presents the total time for evaluating all 50 queries in the scenario, with respect to the initial query selectivity. For brevity, we present results for the MNF dataset on InfluxDB. As the initial selectivity increases, the query interval of each subsequent operation becomes proportionally larger and subsequently, the amount of the data fetched for each query increases. As expected, total evaluation time increases for all competitors as initial selectivity increases. M4 exhibits the largest increase, as each query requires aggregating more data in the database. On the other hand, both MinMaxCache and RawCache demonstrate smaller increases due to the reuse of cached data. However, MinMaxCache outperforms as it is adapting to increased initial query selectivity. Notably, *the evaluation time for MinMaxCache remains within acceptable interactive response times even for high selectivity*, demonstrating its scalability and ability to effectively handle large amounts of data.

Performance vs. Canvas Size. Next, we examine the impact of the canvas size on MinMaxCache and its competitors using the MNF dataset on PostgreSQL (Figure 15). As canvas size increases, the aggregation interval—determined by the aggregation factor (AF) and the pixel column interval, which depends on the visualization canvas's width w in pixels—decreases. This requires retrieving more data points from the database in both our approach and M4, with the latter needing to fetch $4 \times w$ data points for every query. Consequently, larger visualization canvas sizes lead to a

slight increase in query evaluation time for both methods. RawCache performance does not depend on canvas size as it fetches and caches non-aggregated data points. In contrast, OM3's scalability is inferior to our method; notably, at a canvas size of 4000×2400, its performance falls below even that of RawCache. Overall, our method outperforms competitors, even with larger canvas sizes.

5.4 User Evaluation

We conducted a user study to assess MinMaxCache's effectiveness and user acceptance of error-bounded visualizations. Using a prototype visualization tool, 12 participants interacted with a multivariate time series dataset, comparing MinMaxCache-updated charts to error-free charts generated by the M4 approach. The study focused on evaluating interactive performance, the perception of accuracy between error-bounded and error-free visualizations, and confidence in using approximate visualizations for analysis. Tasks were performed at an error bound of 5% and 1%.

Participants favored MinMaxCache for its responsiveness, noting that it enhanced interactivity without significant accuracy loss. Differences between error-bounded and error-free visualizations were subtle and did not affect analytical tasks, such as identifying trends or outliers. Further feedback indicated that tighter error bounds slightly reduced responsiveness without notably improving accuracy. Overall, users expressed willingness to use the error-bounded approach for interactive exploration, suggesting they would resort to error-free visualization for validating selected insights. They also highlighted challenges in understanding the error guarantees, indicating a need for more intuitive presentation of potential errors—issues similarly discussed in prior studies on approximate and incremental visualization [10, 29, 33]. More details about the study and its findings are presented in the paper's supplementary material [5].

6 RELATED WORK

Related work includes data reduction, caching, and approximate query processing techniques over time series data for enhancing interactive visualization performance and accuracy. The challenges of interactivity in the context of large data however, are also encountered in many distinct fields. For example, work in the field of genomics explores challenges associated with interactive searching, browsing, and visualization within large-scale genomic data [6, 28]. In contrast, we focus on dynamic, multivariate time series data, aiming to reduce network latency and ensure visualization accuracy guarantees, thus addressing different analysis goals.

Data Reduction. Data reduction techniques [17], [13] can improve visualization latency, by transferring sampled, aggregated, or model-based data subsets to the front-end. Such methods, however, can distort the original shape of the data. M4 [15] adopts a pixel-oriented strategy, preserving the shape of the line chart, while reducing latency. Yet, M4 lacks support for reusing results for operations like panning and zooming, which can affect interactivity with large time series data. OM3 [36] proposes a pre-computed, multi-level min-max representation for progressive visualization. Besides the preprocessing phase, its limitations include large disk space requirements, and no support for updates and streaming data. Line simplification techniques, [8, 11, 35] are effective for visualization accuracy, but computationally intensive for large datasets.

Several works propose memory-oriented variations[20, 21, 31] of the data cube structure [12] for interactive visualization, but face challenges like high memory consumption, lengthy preprocessing, and inadequate support for data updates and streaming data. A time series-specific variation is presented in [27], addressing updates and streaming data, and adopting a temporal hierarchy-based materialization strategy. However, it depends on predefined calendar-based aggregation levels (e.g., month, day) and cannot represent raw data with satisfactory accuracy. Similarly, [34] suggests a calendar-based multi-level aggregation structure for visualizing solar panel data.

In another context, memory-based indexing approaches for interactive exploration of raw data files are introduced in [3, 4, 24–26, 32]. Despite dynamically determining data granularity for stored aggregated metadata, these methods are predominant in 2D scenarios.

Caching. Caching has been widely studied to reduce database latency. Works like [38] and TSCache [22] propose time-range-based caching for time series data to improve query execution time by reusing cached results. Unlike our work, these generic approaches overlook the unique challenges specific to visualization of time series data and do not incorporate data aggregation in order to reduce query latency while adjusting the aggregation granularity to maintain an acceptable accuracy in the final visualization.

Approximate Query Processing (AQP). Various AQP-based approaches have been proposed for faster visualization interaction while computing precise results[7, 10, 18, 29, 33]. Our approach differs by considering the mapping from value to pixel space and defining an error bound in terms of pixel differences relative to accurate visualization. While some works consider human perception [1, 18], their objectives differ from ours. [18] focuses on quickly generating approximate bar charts, preserving pairwise bar ordering, while [1] proposes an online sampling algorithm that learns human perception error to provide visualizations with perceptually indiscernible error. Our work on the other hand employs pixel-aware aggregation to guarantee the accuracy constraints and allow the reuse of cached data to minimize query latency.

7 CONCLUSIONS

In this paper, we presented *MinMaxCache*, an adaptive caching approach for interactive exploration of large time series data, adhering to visualization accuracy constraints. Our approach leverages cached min-max aggregates of time series intervals at dynamically adjusted granularities. Upon receiving a query, *MinMaxCache* computes a visualization error bound using cached data and fetches additional data at appropriate granularity to balance accuracy and response times. Evaluations indicate that our method outperforms existing solutions, providing up to 10 times faster query evaluation.

ACKNOWLEDGMENTS

This work was partially supported by ExtremeXP project (EU Horizon program, GA 101093164) and MORE project (EU H2020 program, GA 957345).

[5] https://github.com/athenarc/MinMaxCache/blob/main/paper_supplementary_material.pdf

REFERENCES

[1] Daniel Alabi and Eugene Wu. 2016. Pfunk-h: Approximate query processing using perceptual models. In *Proceedings of the workshop on human-in-the-loop data analytics*. 1–6.

[2] Leilani Battle, Remco Chang, and Michael Stonebraker. 2016. Dynamic prefetching of data tiles for interactive visualization. In *Proceedings of the 2016 International Conference on Management of Data*. 1363–1375.

[3] Nikos Bikakis, Stavros Maroulis, George Papastefanatos, and Panos Vassiliadis. 2018. RawVis: visual exploration over raw data. In *Advances in Databases and Information Systems: 22nd European Conference, ADBIS 2018, Budapest, Hungary, September 2–5, 2018, Proceedings 22*. Springer, 50–65.

[4] Nikos Bikakis, Stavros Maroulis, George Papastefanatos, and Panos Vassiliadis. 2021. In-situ visual exploration over big raw data. *Information Systems* 95 (2021), 101616.

[5] Nikos Bikakis, George Papastefanatos, Melina Skourla, and Timos Sellis. 2017. A hierarchical aggregation framework for efficient multilevel visual exploration and analysis. *Semantic Web* 8, 1 (2017), 139–179.

[6] Arnaud Ceol, Piero Montanari, Ilaria Bartolini, Stefano Ceri, Paolo Ciaccia, Marco Patella, and Marco Masseroli. 2020. Search and comparison of (epi) genomic feature patterns in multiple genome browser tracks. *BMC bioinformatics* 21, 1 (2020), 1–13.

[7] 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.

[8] 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.

[9] Herbert Edelsbrunner. 1980. *Dynamic data structures for orthogonal intersection queries*. Technische Universität Graz.

[10] Danyel Fisher, Igor Popov, Steven Drucker, and MC Schraefel. 2012. Trust me, I'm partially right: incremental visualization lets analysts explore large datasets faster. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 1673–1682.

[11] Tak-chung Fu, Fu-lai Chung, Robert Luk, and Chak-man Ng. 2008. Representing financial time series based on data point importance. *Engineering Applications of Artificial Intelligence* 21, 2 (2008), 277–300.

[12] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. 1997. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery* 1 (1997), 29–53.

[13] Søren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. 2021. Scalable Model-Based Management of Correlated Dimensional Time Series in ModelarDB+. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. 1380–1391. https://doi.org/10.1109/ICDE51399.2021.00123

[14] Z. Jerzak, T. Heinze, M. Fehr, D. Grober, R. Hartung, and N. Stojanovic. 2012. The DEBS 2012 Grand Challenge. *DEBS* (2012), 393–398. https://debs.org/grand-challenges/2012/

[15] Uwe Jugel, Zbigniew Jerzak, Gregor Hackenbroich, and Volker Markl. 2014. M4: a visualization-oriented time series data aggregation. *Proceedings of the VLDB Endowment* 7, 10 (2014), 797–808.

[16] Alexander Kalinin, Ugur Cetintemel, and Stan Zdonik. 2014. Interactive data exploration using semantic windows. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 505–516.

[17] Eamonn J Keogh and Michael J Pazzani. 2000. A simple dimensionality reduction technique for fast similarity search in large time series databases. In *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 122–133.

[18] Albert Kim, Eric Blais, Aditya Parameswaran, Piotr Indyk, Sam Madden, and Ronitt Rubinfeld. 2015. Rapid sampling for visualizations with ordering guarantees. In *Proceedings of the vldb endowment international conference on very large data bases*, Vol. 8. NIH Public Access, 521.

[19] Intel Berkeley Research Lab. 2004. Intel Lab Dataset. (2004). http://db.csail.mit.edu/labdata/labdata.html

[20] Lauro Lins, James T Klosowski, and Carlos Scheidegger. 2013. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2456–2465.

[21] Can Liu, Cong Wu, Hanning Shao, and Xiaoru Yuan. 2019. Smartcube: An adaptive data management architecture for the real-time visualization of spatiotemporal datasets. *IEEE transactions on visualization and computer graphics* 26, 1 (2019), 790–799.

[22] Jian Liu, Kefei Wang, and Feng Chen. 2021. TSCache: An Efficient Flash-Based Caching Scheme for Time-Series Data Workloads. *Proc. VLDB Endow.* 14, 13 (sep 2021), 3253–3266. https://doi.org/10.14778/3484224.3484225

[23] Zhicheng Liu and Jeffrey Heer. 2014. The effects of interactive latency on exploratory visual analysis. *IEEE transactions on visualization and computer graphics* 20, 12 (2014), 2122–2131.

[24] Stavros Maroulis, Nikos Bikakis, George Papastefanatos, Panos Vassiliadis, and Yannis Vassiliou. 2021. Adaptive Indexing for In-situ Visual Exploration and Analytics.. In *DOLAP*. 91–100.

[25] Stavros Maroulis, Nikos Bikakis, George Papastefanatos, Panos Vassiliadis, and Yannis Vassiliou. 2021. RawVis: A System for Efficient In-Situ Visual Analytics. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 2760–2764. https://doi.org/10.1145/3448016.3452764

[26] Stavros Maroulis, Nikos Bikakis, George Papastefanatos, Panos Vassiliadis, and Yannis Vassiliou. 2023. Resource-aware adaptive indexing for in situ visual exploration and analytics. *The VLDB Journal* 32, 1 (2023), 199–227.

[27] Fabio Miranda, Marcos Lage, Harish Doraiswamy, Charlie Mydlarz, Justin Salamon, Yitzchak Lockerman, Juliana Freire, and Claudio T Silva. 2018. Time lattice: A data structure for the interactive visual analysis of large time series. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 23–35.

[28] Piero Montanari, Ilaria Bartolini, Paolo Ciaccia, Marco Patella, Stefano Ceri, and Marco Masseroli. 2016. Pattern similarity search in genomic sequences. *IEEE Transactions on Knowledge and Data Engineering* 28, 11 (2016), 3053–3067.

[29] 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 2017 CHI conference on human factors in computing systems*. 2904–2915.

[30] C. Mutschler, H. Ziekow, and Z. Jerzak. 2013. The DEBS 2013 Grand Challenge. *DEBS* (2013), 289–294. https://debs.org/grand-challenges/2013/

[31] Cicero AL Pahins, Sean A Stephens, Carlos Scheidegger, and Joao LD Comba. 2016. Hashedcubes: Simple, low memory, real-time visual exploration of big data. *IEEE transactions on visualization and computer graphics* 23, 1 (2016), 671–680.

[32] George Papastefanatos, Giorgos Alexiou, Nikos Bikakis, Stavros Maroulis, and Vassilis Stamatopoulos. 2022. VisualFacts: A Platform for In-Situ Visual Exploration and Real-Time Entity Resolution.. In *EDBT/ICDT Workshops*.

[33] Sajjadur Rahman, Maryam Aliakbarpour, Ha Kyung Kong, Eric Blais, Karrie Karahalios, Aditya Parameswaran, and Ronitt Rubinfield. 2017. I've seen" enough" incrementally improving visualizations to support rapid decision making. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1262–1273.

[34] Vassilis Stamatopoulos, Stavros Maroulis, Konstantinos Kozanis, Ioannis Psarros, George Papastefanatos, Giorgos Giannopoulos, and Manolis Terrovitis. 2023. A Tool for Visual Exploration and Analysis of Solar Photovoltaic Module Data. (2023).

[35] Maheswari Visvalingam and James D Whyatt. 1993. Line generalisation by repeated elimination of points. *The cartographic journal* 30, 1 (1993), 46–51.

[36] 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. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–24.

[37] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13, 4 (2004), 600–612.

[38] Thada Wangthammang and Pichaya Tandayya. 2018. A software cache mechanism for reducing the OpenTSDB query time. In *2018 18th International Symposium on Communications and Information Technologies (ISCIT)*. IEEE, 60–65.