



Department of Management Science & Technology

MSc in Business Analytics

Approximate Visual Exploration of Large Time Series Data with Accuracy Guarantees

By

Christos Pantoleon

Student ID Number: p2822219

Name of Supervisor: Papastefanatos Georgios

July 2025

Athens, Greece



Table of contents

1. Introduction.....	1
2. Related work.....	5
2.1. Visualization methods.....	6
2.1.1. M4.....	6
2.1.2. OM3.....	6
2.1.3. MinMaxCache.....	7
2.2. Benchmarking tools.....	7
2.2.1. TPC-H.....	7
2.2.2. TPC-DS.....	8
2.2.3. Star Schema Benchmark (SSB).....	9
2.2.4. TSM-Bench.....	10
2.2.5. IDEBench.....	11
2.2.6. Simulation-Based Benchmark (SIMBA).....	11
3. Methodology.....	12
3.1. TimeVizBench Front End Architecture.....	13
3.1.1. Client-Server Interaction Flow.....	15
3.1.2. Handling User Interaction.....	21
3.1.3. Rendering Lines with Pixel Precision.....	25
3.1.4. Evaluating Visualization Method Performance.....	29
3.2. User Interface.....	30
3.2.1. UI Components.....	31
3.2.1.1. Configuration Panel.....	33
3.2.1.2. Visualization Panel.....	35
3.2.1.3. Performance Metrics Panel.....	36
3.3. Adding a visualization method.....	38
4. Conclusions and Future Work.....	39
4.1. Conclusions.....	39
4.2. Future Work.....	41
5. Acknowledgements.....	41
References.....	42



Abstract

The exponential growth of time series data in both the size and complexity has presented notable obstacles for developing visualizations that are simultaneously precise and efficient. Balancing high visual fidelity with minimal latency remains a persistent challenge. Visualization plays a critical role in time series data exploration, enabling users to detect trends, patterns, and anomalies that may not be apparent from raw data alone.

While data reduction strategies such as M4 and MinMaxCache are intended to decrease response times while preserving visual quality, there remains a distinct absence of straightforward, practical frameworks for assessing their effectiveness in real-world contexts. Existing benchmarks typically emphasize backend efficiency or user interaction. However, they often fall short in evaluating the dual concerns of visualization accuracy and system responsiveness, issues that become increasingly difficult to manage as dataset size and complexity grow.

This thesis presents TimeVizBench, a web-based benchmarking platform designed to systematically assess and compare time series visualization methods with respect to both latency and fidelity. The system enables users to interact with visualizations through an accessible interface, and provides performance metrics, including backend query time, frontend rendering duration, and data transfer latency.

For the measurement of visual accuracy, the tool uses the Structural Similarity Index (SSIM), offering an objective metric for comparing rendered visualizations. Leveraging pixel-precise rendering via D3.js and supporting the integration of custom visualization techniques, TimeVizBench provides a robust, extensible framework for researchers and practitioners aiming to evaluate visualization strategies under real-world conditions.

1. Introduction

Time series data consists of observations collected sequentially over time, where each data point is associated with a specific timestamp and a value. This type of data is prevalent across various domains, including finance, healthcare, energy, and industrial monitoring. As data acquisition becomes increasingly automated and granular, the volume of time series data being generated has grown dramatically. These datasets are typically stored in specialized time-series databases (TSDBs) such as InfluxDB or TimescaleDB, or in general-purpose relational databases. However, analyzing time series data in its raw tabular form can be challenging, especially when it spans millions of entries. In such scenarios, essential trends, seasonal patterns, or anomalies often remain hidden or are too subtle to detect without visualization.

Visualization plays a critical role in enabling users to comprehend large-scale time series datasets. It transforms raw, high-frequency data into an intuitive visual format that supports pattern recognition, correlation detection, and decision-making. For instance, the upward trend of a stock market index such as the S&P 500 can be immediately recognized in a line chart but remains obscure when inspecting a numerical table. Figure 1.1 shows raw tabular data of S&P 500 prices over time, while Figure 1.2 visualizes the same dataset, clearly revealing long-term trends and short-term volatility. This contrast highlights the importance of visual representation as a bridge between raw data and human understanding.

SP500

observation_date	SP500
2020-03-09	2746.56
2020-03-10	2882.23
2020-03-11	2741.38
2020-03-12	2480.64
2020-03-13	2711.02

Figure 1.1. The price of S&P 500

Source: S&P Dow Jones Indices LLC via FRED®



Figure 1.2. Visualization of the time series data like the ones on Figure 1.

Source: S&P Dow Jones Indices LLC via FRED®

The quality of the interpretation of the data depends on the accuracy of the representation. Accuracy, in this context, refers to the precise mapping of values to visual elements on the output device without distortion, misalignment, or loss of critical details. Additionally, it involves minimizing perceptual biases caused by improper axis scaling, or inadequate differentiation of data points. An accurate visualization should enable users to derive correct insights without causing misinterpretation due to display limitations or graphical inconsistencies.

Creating accurate and responsive visualizations at scale introduces significant technical challenges. As the resolution of output devices is finite, attempting to render millions of data points onto a screen with only thousands of pixels leads to overplotting and visual artifacts. A standard full-HD screen with 1920 horizontal pixels cannot visually distinguish between points spaced at subpixel distances. Consequently, visualizing raw time series data often results in either visual clutter or loss of detail due to aggressive downsampling.

Data visualization also depends on retrieving the data points from storage. Data retrieval is a process including loading data from a data source, processing them, transmitting them over the network, and finally rendering them. Depending on the processing required in each case i.e. performing aggregation on the database, post-processing the result etc. each step introduces



latency that affects how often users can query new data. The latency of the overall system is correlated with the amount of data that is processed, which means that a large dataset will take more time to be processed than a smaller one. Latency affects the interactivity of the visualization which is the ability of the user to effectively navigate in the graphical representation of a dataset, and manifests as the total delay between data retrievals.

The increasing volume of large-scale time series data poses a challenge when trying to effectively visualize them with 100% accuracy while maintaining low latency. The large volume of data affects the accurate graphical representation due to the physical limitations of the output devices i.e. the amount of available pixels[1] which may lead to over-plotting and ultimately to loss of information. Additionally, trying to meet low latency requirements, require excessive processing power and additional storage.

To address these challenges, data reduction techniques were developed that aim to maximize accuracy while minimizing latency. Such technique is M4, an aggregation-based time series dimensionality reduction technique. While traditional techniques focus on reducing the volume of data, ignoring the semantics of visualizations thus lowering the accuracy, M4 introduces a visualization-aware solution where it reduces the volume of data needed by aggregating data points that their timestamps are close enough to be rendered in the same pixel column, achieving 100% accuracy and lower latency when comparing with the visualization of the raw dataset[2].

An alternative data reduction technique is MinMaxCache, a visualization-aware reduction technique that further reduces latency by calculating and caching data aggregations with time ranges smaller than the time range of a single pixel column that can be used in consequent queries. The reusability of the cached data enhances the interactivity of the chart by loading from the system's memory cached data from previous queries while only missing data are loaded from the data source instead of aggregating all the rows for the requested time range. The added benefit of the swifter response is the introduction of visualization errors.

Selecting the appropriate visualization method boils down to the latency and accuracy requirements of each application. To make an informed decision, the trade-off should be easily identifiable and the performance gains of each method quantifiable. Therefore, benchmarking is



crucial to assess the trade-offs between accuracy and performance across different reduction techniques.

State of the art benchmarking methodologies focus on measuring backend performance of data warehouses or specifically query engines handling time series data, failing to evaluate the interactivity and accuracy, while visualization-specific benchmarks don't focus on handling large scale time series data. Additionally, benchmarks that evaluate system-level performance for exploration and visualization, fail to measure the trade-off between accuracy and usability in terms of latency.

To fill this gap, this thesis introduces TimeVizBench [16], a tool designed to compare and evaluate different visualization methods for large-scale time series through an interactive, user-friendly UI, providing insights about the performance of each method in terms of accuracy and latency. This tool enables the user to set up the comparison testbed by selecting datasets, measures included in those datasets, visualization methods with their respective instantiation parameters, and interacting with the rendered line chart.

Latency is measured as the total end-to-end time from initiating a request to rendering the chart, broken down into network latency, backend processing time, and frontend rendering time. Accuracy is assessed using Structural Similarity index (SSIM), a perceptual image similarity metric that compares the rendered chart to a reference visualization based on raw data. SSIM captures visual structure rather than pixel-by-pixel differences, aligning well with human perception.

This tool provides both researchers and practitioners with the means to make informed decisions when selecting visualization methods, especially in environments where performance and correctness are both critical.

Later in this document Section 2 provides an overview of the related work done in the field and the state of the art. Section 3 describes our methodology on approaching this problem. Section 4 provides the evaluation of our tool and a walkthrough of how users can interact with it, and, finally, Section 5 concludes the thesis providing future work and lessons learned.



2. Related work

This section reviews state of the art benchmarking frameworks and large-scale time series visualization techniques, considering their objectives, data generation methods, evaluation criteria, and overall significance to the current research.

Benchmarks such as TPC-H, TPC-DS, SSB, and TSM-Bench are discussed for their strategies involving either synthetic or scaled real-world datasets. Their primary focus tends to be on backend performance, metrics like query speed, throughput, and system resource usage dominate the discussion. While these benchmarks are effective for assessing database infrastructure, they largely overlook how end-users actually interact with query results, particularly in terms of visualization and interpretation. Essentially, they offer a robust measure of technical performance but provide minimal insight into user-facing aspects like data consumption or visualization workflows.

More recent benchmarks such as IDEBench and SIMBA introduce more interactive and user-centered evaluation strategies, incorporating simulated exploration tasks and sequences of user interactions. These tools often include their own data generators to seed the system under test with realistic datasets, and they assess factors like system responsiveness, interactivity, and in some cases, partial correctness of intermediate results. However, these benchmarks overlook visualization accuracy, usability, and cognitive effectiveness.

By reviewing how current benchmarks handle data seeding, simulate user behavior, and define their evaluation scope, this section identifies critical gaps in the ability to measure visualization effectiveness. These limitations motivate the development of the proposed methodology, which aims to offer a more comprehensive framework that assesses not only performance and interactivity but also the visual clarity, accuracy, and interpretability of different time series visualization techniques.



2.1. Visualization methods

2.1.1. M4

M4 is a visualization-aware time series aggregation method [2] that aims to preserve the visual fidelity of large time series when displayed in line charts. It transforms user queries into pixel-aware aggregation queries that compute the minimum and maximum values for each vertical pixel column in the display. This ensures that every visual feature in the original dataset is captured without distortion or omission. By aligning its aggregation with the resolution of the user's screen, M4 significantly reduces the amount of data transferred and rendered, often by two orders of magnitude, while maintaining an error-free visual output.

The M4 approach is implemented as a query rewriting technique within a traditional RDBMS, allowing for integration without modifying the database engine. However, one notable drawback is its impact on interactivity. Since every new viewport—such as a zoom or pan—requires a new query to be issued and processed, response times can suffer, especially on large datasets. This latency makes M4 more suitable for static or exploratory use cases rather than continuous, real-time interaction.

2.1.2. OM3

OM3 is a multiresolution method [15] for interactive visualization that precomputes and stores min-max aggregates across multiple levels of granularity. It organizes these aggregates in a tree-like hierarchy, enabling the system to progressively refine visual output as the user interacts with the data. This design allows OM3 to support continuous interactions such as zooming and panning without recomputing data from scratch for every new view.

The method operates through a forward transform that builds successive levels of aggregation and a tree-based query algorithm that incrementally provides data for rendering. Because OM3 relies on precomputed results, it can maintain a smooth user experience during interactive exploration. Its primary trade-offs lie in the need for offline preprocessing and additional storage to maintain the multilevel hierarchy.



2.1.3. MinMaxCache

MinMaxCache is a visualization-aware caching strategy [14] that extends pixel-aligned aggregation methods by storing min-max summaries from past queries and reusing them for future ones. It uses user-defined error bounds to determine whether cached data is sufficiently accurate to render a new viewport, avoiding the need to recompute aggregations if the visual error remains within the specified threshold.

This approach supports fast and flexible visual exploration by leveraging existing aggregates instead of always querying the full dataset. The cache operates at multiple granularities and is designed to balance performance with visual accuracy. While effective for interactive use, MinMaxCache assumes univariate time series data and requires careful management of cache memory and error thresholds.

An applied solution that leverages the functionality of MinMaxCache similar to TimeVizBench was developed [17] that provided a web-based graphical interface enabling the user to explore data generated by sensors in solar photovoltaic installations to monitor and optimize their performance.

2.2. Benchmarking tools

2.2.1. TPC-H

TPC-H is a state-of-the-art benchmark developed by the Transaction Processing Performance Council (TPC) to evaluate the performance of Decision Support Systems (DSS)[3]. These systems are widely used in business intelligence contexts to analyze large volumes of structured data, identify patterns, generate analytical reports, and provide insights that inform strategic decision-making. A typical DSS executes complex, resource-intensive queries that include operations such as aggregations, filtering, multi-table joins, subqueries, and computational expressions. These queries are often exploratory and ad hoc in nature, reflecting real-world scenarios where users iteratively probe datasets to uncover insights.



The TPC-H benchmark consists of a suite of 22 business-oriented ad hoc queries and a set of concurrent data modification operations. These queries simulate the interaction between a user and a DSS application by posing questions such as forecasting revenue, analyzing supplier performance, or detecting trends in customer orders. To ensure consistency and scalability across different database systems, TPC-H specifies detailed data schemas and population requirements that define the size and distribution of the data. This allows performance to be tested at various data scales, from gigabytes to terabytes.

While TPC-H is a gold standard for evaluating backend database performance, including aspects like query execution time, throughput, and system resource utilization, it does not assess how query results are presented or interpreted by end-users. Specifically, it provides no mechanisms to evaluate the accuracy, clarity, or usability of visual representations of the data. In modern analytics workflows, especially those involving large-scale time series data, visualization plays a central role in making sense of complex results. However, TPC-H assumes that query output is consumed passively, and does not account for how visualization tools or user interfaces mediate data interpretation. This limitation makes it insufficient for evaluating systems where user interactivity and visual sense-making are core components of the analytic experience.

2.2.2. TPC-DS

TPC-DS is another widely adopted benchmark developed by the Transaction Processing Performance Council, designed to evaluate the performance of Decision Support Systems (DSS) in the context of complex and realistic business intelligence workloads[4]. Like TPC-H, it focuses on measuring how well database systems handle analytical queries; however, TPC-DS reflects a more modern and sophisticated view of decision support, making it particularly relevant for contemporary data warehouse environments.

One of the key advancements of TPC-DS over TPC-H lies in the complexity and richness of its query workload. It features 99 SQL queries that cover a broader range of operations, including window functions, nested subqueries, roll-ups, advanced grouping, and common table expressions, constructs that are increasingly important in real-world analytics scenarios. The queries simulate a multi-dimensional decision-making environment, such as that found in a retail



business, with analyses involving sales forecasting, inventory tracking, customer behavior, and promotion effectiveness.

TPC-DS also introduces a more complex schema compared to TPC-H. Instead of using a single flat star schema, it models a snowflake schema with multiple interrelated dimension tables and fact tables, better reflecting the structure of modern enterprise data warehouses. Additionally, TPC-DS includes both reporting and interactive query types, as well as ETL components to simulate the extraction, transformation, and loading of data, allowing it to benchmark a broader portion of the analytics pipeline.

Despite these enhancements, TPC-DS remains focused exclusively on backend performance metrics, such as query execution time, throughput, and system scalability. It does not evaluate how query results are presented, interpreted, or used by end-users. In particular, it does not address the role of visualization in the analytic process, an increasingly vital aspect of modern DSS environments, where users rely on dynamic dashboards and interactive visualizations to explore data and derive insights. As such, while TPC-DS provides a rigorous framework for benchmarking the technical capabilities of DSS backends, it does not offer any guidance on how effectively these systems support user-centric, visually driven data exploration.

2.2.3. Star Schema Benchmark (SSB)

The Star Schema Benchmark (SSB) is a benchmarking framework developed to assess the performance of analytical queries on relational database systems, particularly in the context of data warehousing workloads. It is derived from the TPC-H benchmark but introduces several modifications aimed at improving relevance and efficiency for data warehouse evaluation. The most significant change is its adoption of a star schema, a widely used schema design in data warehouses, rather than the normalized schema used in TPC-H.

The star schema simplifies query patterns and enhances performance by reducing the number of joins required during query execution. This makes SSB more representative of the types of workloads encountered in real-world data warehousing scenarios. The benchmark includes a suite of parameterized queries designed to evaluate how well a database system can optimize and



execute decision support queries at various levels of complexity and data scale. A key focus of SSB is the assessment of query optimizer effectiveness, particularly in generating efficient execution plans under different workload configurations[5].

However, similar to TPC-H and TPC-DS, the scope of SSB is limited to backend performance. It does not account for how query results are visualized, interpreted, or used by end-users. As a result, it does not provide any insights into the usability or accuracy of visualization methods, which are increasingly critical components in analytical workflows, especially when dealing with complex or high-volume data such as time series.

2.2.4. TSM-Bench

TSM-Bench is a benchmarking framework specifically developed to evaluate the performance of time series database systems (TSDBs)[11]. Unlike general-purpose relational database benchmarks such as TPC-H, TPC-DS, and SSB, which are optimized for traditional decision support systems, TSM-Bench focuses on the characteristics and demands of time series data. Time series data is high-volume, high-frequency, and time-ordered, requiring specialized data structures and query processing techniques.

To simulate realistic workloads, TSM-Bench includes a data generation component that produces synthetic datasets based on real-world patterns, such as periodic measurements, event spikes, and continuous streams. It also defines a set of representative queries that perform complex operations like aggregations over time windows, anomaly detection, or statistical summarization. These workloads are designed to stress the ingestion pipeline, storage engine, and query processor, allowing for meaningful comparisons between TSDB implementations. The benchmark supports variable dataset sizes and configurable query templates, making it adaptable to different scales and deployment scenarios.

Despite its strengths in evaluating system performance, including query latency, throughput, ingestion rate, and storage efficiency, TSM-Bench, like many traditional database benchmarks, does not provide insights about end-user experience. It provides no means of assessing how query results are presented, explored, or interpreted through visualization tools.



2.2.5. IDEBench

IDEBench is a benchmarking tool designed to assess the performance of interactive data exploration systems, particularly in contexts where users generate queries in an ad hoc and iterative manner rather than adhering to predetermined workloads[12]. This approach reflects how data is explored in real-world scenarios, where individuals continuously refine their queries based on real-time system feedback. Notably, IDEBench incorporates a data generation module capable of populating databases with synthetic data modeled on real-world distributions. This feature supports reliable, reproducible evaluations, enhancing the validity of performance comparisons across different systems.

Unlike traditional benchmarks like TPC-H, which rely on a predetermined set of static queries to represent reporting workloads, IDEBench adopts a more realistic approach by simulating interactive user sessions. These sessions involve typical data exploration behaviors, like filtering, grouping, aggregating, and drilling down, that mirror how individuals actually interact with analytical dashboards and interfaces. This methodology moves the emphasis beyond just raw query-processing speed, instead highlighting the importance of system responsiveness and usability from the perspective of end users.

While IDEBench provides a valuable framework for evaluating interactivity and user-driven exploration, it does not specifically address time series data.

2.2.6. Simulation-Based Benchmark (SIMBA)

The Simulation-Based Benchmark (SIMBA) is a benchmarking framework developed to assess the performance of database management systems (DBMSs) in the context of interactive data exploration through dashboards[13]. Unlike traditional benchmarks that primarily measure backend query performance using predefined workloads, SIMBA focuses on simulating realistic user interactions and analysis goals within visual dashboard environments. Its key innovation lies in modeling analytical tasks as sequences of user-driven SQL queries, which are derived from valid interactions a user might perform while navigating and exploring a dashboard interface.



By translating high-level user goals into executable queries, SIMBA enables a more behaviorally grounded evaluation of how a system supports interactive exploration. It captures important performance metrics such as query response times and data accuracy, particularly in relation to how closely the system's responses align with expected outcomes during exploration sessions.

Despite its contributions, SIMBA has several limitations when considered in the context of large-scale time series analysis. It is not specifically designed to handle high-frequency, time series data, and lacks the mechanisms required to evaluate how systems support their interactive, multiscale exploration. Additionally, the benchmark itself does not provide a live interactive environment for real users to engage with, which limits its ability to fully capture user experience factors such as interface responsiveness or visual clarity during exploration.

3. Methodology

This thesis introduces TimeVizBench, a web-based benchmarking tool specifically designed to evaluate and compare various visualization techniques for large-scale time series data. The system architecture consists of two components: a frontend Single Page Application (SPA) and a backend API service. The frontend, written in JavaScript using React.js, serves as the user interface, enabling users to interact with datasets, adjust visualization parameters, and access both performance metrics and generated charts. Its single-page structure promotes a responsive and uninterrupted user experience, providing dynamic content updates without necessitating full-page reloads.

The backend is implemented as a Java-based REST API service using SpringBoot. It handles requests from the frontend, including metadata retrieval, time series query execution, and the application of selected visualization methods. The backend also manages each method's initialization parameters, performs server-side data aggregation or caching, and ensures efficient data delivery for rendering.

This separation of concerns between frontend and backend allows TimeVizBench to remain modular, extensible, and efficient when dealing with high-volume, high-frequency time series data. By providing a clear structure for comparing latency and accuracy across multiple

visualization methods, the tool supports both research-driven evaluation and practical decision-making in the development of scalable visual analytics systems.

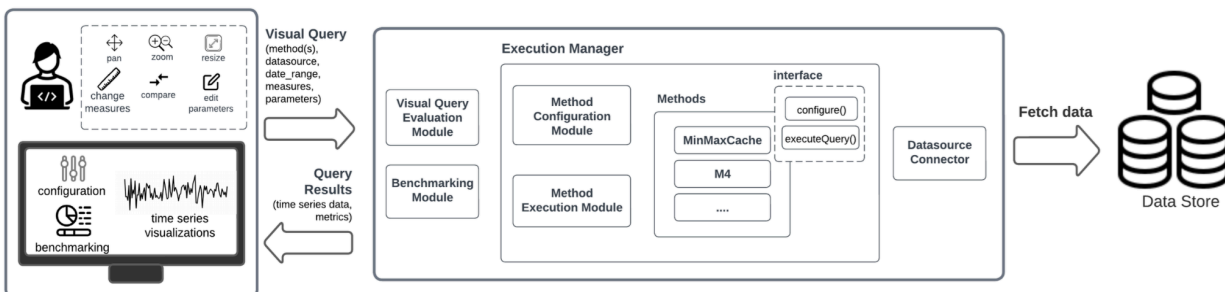


Figure 3.1. TimeVizBench high level architecture diagram.

3.1. TimeVizBench Front End Architecture

The SPA is composed of three main layers: the UI layer, which contains the components rendered on screen; the state management layer, which handles application-wide state; and the data layer, which manages communication with the backend API.

The UI layer is built using ReactJS, a declarative JavaScript library that enables the creation of reusable and self-contained components. Each major element of the interface, such as the dataset selector, visualization configuration panel, chart display, and performance metrics viewer, is implemented as an independent component. React's virtual DOM diffing ensures efficient re-rendering of the interface in response to user interactions and data changes, which is essential for maintaining responsiveness during dynamic time series exploration. The modular component structure also makes it easier to introduce future enhancements, such as reconfigurable layouts or additional interactive widgets.

For rendering time series visualizations, the application uses D3.js (Data-Driven Documents), a powerful library designed to produce dynamic and interactive visualizations by binding data directly to the Document Object Model (DOM). While WebGL and Canvas are often used for high-performance rendering of large datasets, D3.js was chosen for its fine-grained control over SVG elements, which is essential for achieving pixel-perfect rendering. Unlike abstract charting



libraries or raster-based approaches, D3 provides low-level control over scales, axes, and paths allowing precise manipulation of rendering behavior in relation to screen resolution and device pixel ratio.

This level of precision is critical for benchmarking visualization accuracy, where even minor misalignments can significantly affect SSIM-based evaluations. Additionally, D3's declarative style integrates well with React, making it well-suited for building an interactive and visually consistent benchmarking tool like TimeVizBench.

Asynchronous communication with the backend is managed through Axios, a promise-based HTTP client. The frontend issues requests for dataset metadata, available visualization methods, and rendered chart data. These requests are parameterized according to the user's current configuration and include mechanisms for cancellation and timeout, ensuring that outdated or duplicate requests are cancelled. This is particularly important in interactive contexts where rapid changes to visualization settings, such as zooming or method switching, can generate overlapping queries. The frontend uses an AbortController to manage this interaction flow, improving both accuracy and responsiveness.

Finally, state management in the frontend is handled using the React-Redux library, which provides a predictable and centralized approach to managing application state. Redux maintains a single global state tree, allowing components across the application to access and update shared data in a consistent and traceable manner.

3.1.1. Client-Server Interaction Flow

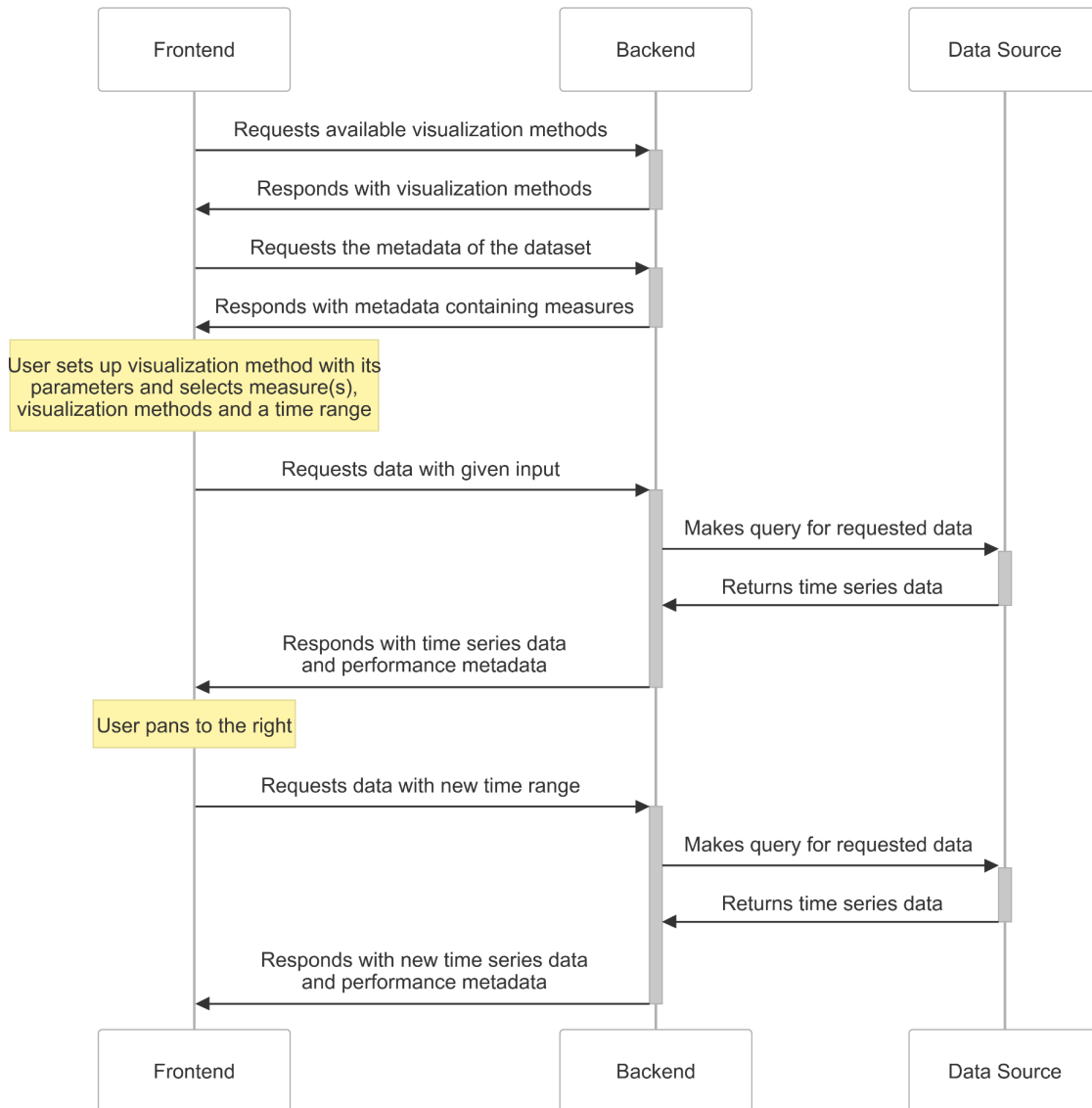


Figure 3.4. The sequence diagram of the user interaction. The method configuration response contains the names of the methods and their initialization parameters.

Loading and Presenting Visualization Method Metadata. The visualization methods metadata are loaded first to enable the user to select one or more of the available methods from the “Method” drop-down list. The metadata contains the names of the methods, their descriptions, and their respective initialization parameters. For example, the method "MinMaxCache" is



described as a “cache-based visualization method using min-max aggregates”, and it includes three configurable initialization parameters: aggregation factor, data reduction ratio, and prefetching factor. Each parameter is defined with a label (e.g., "Aggregation Factor"), a default value, a range (minimum and maximum), a step size (e.g., 2 or 0.1), and a description explaining its role such as controlling the aggregation level or data reduction ratio. Query-time parameters may also be included, like accuracy which define how the visualization should behave at runtime. These metadata such as type, range, and labels to ensure input validation and user guidance in the frontend interface.

In contrast, the "M4" method, described as a “raw data representation method that uses reduction techniques to visualize large datasets”, does not require any initialization parameters, making it simpler to configure. Some methods, such as MinMaxCache.

```

1 {
2   "M4": {
3     "initParams": {},
4     "queryParams": {},
5     "description": "Raw data representation method that uses reduction techniques to visualize large datasets"
6   },
7   "MinMaxCache": {
8     "initParams": {
9       "prefetchingFactor": {
10        "default": 0,
11        "min": 0,
12        "max": 1,
13        "description": "Controls the prefetching amount (Float)",
14        "step": 0.1,
15        "label": "Prefetching Factor",
16        "type": "number"
17      },
18      "aggFactor": {
19        "default": 4,
20        "min": 2,
21        "max": 16,
22        "description": "Controls the aggregation level (Integer)",
23        "step": 2,
24        "label": "Aggregation Factor",
25        "type": "number"
26      },
27      "dataReductionRatio": {
28        "default": 6,
29        "min": 0,
30        "max": 12,
31        "description": "Controls the data reduction ratio (Integer)",
32        "step": 1,
33        "label": "Data Reduction Factor",
34        "type": "number"
35      }
36    },
37     "queryParams": {
38       "accuracy": {
39        "default": 0.95,
40        "min": 0,
41        "max": 1,
42        "description": "Controls the accuracy of results (Float)",
43        "step": 0.01,
44        "label": "Accuracy",
45        "type": "number"
46      }
47     },
48     "description": "Cache-based visualization method using min-max aggregates"
49   }
50 }
51

```

Figure 3.5. The method configuration response contains the names of the methods and their initialization parameters.

Interacting with the chart. When a user selects a datasource within the TimeVizBench interface, the system initiates a network request to the backend in order to retrieve relevant metadata about the selected dataset. The response includes key descriptive information such as the temporal extent of the dataset, specifically the earliest (start) and latest (end) timestamps available. This information is used to define the valid bounds for selecting a time range for visualization and ensures that users do not attempt to query outside the dataset's available window.



In addition to the time range, the metadata also includes a list of available measures, which represent the quantitative variables or dimensions recorded in the dataset. In the metadata response, these are listed under the "measures" field as an array of numerical identifiers (e.g., [0, 1, 2, 3, 4, 5, 6]), each corresponding to a data column. These measures are associated with the actual column names provided in the "header" array (e.g., "value_1", "value_2", ..., "value_7"), which are displayed in the user interface as selectable options. This allows users to choose one or more dimensions to visualize and compare.

By decoupling the data content from the initial application load and dynamically fetching it as needed, this metadata-driven approach improves efficiency, enables dataset flexibility, and supports a more interactive user experience. Other metadata fields, such as "schema", "tableName", and "samplingInterval", provide additional contextual information that ensures proper dataset handling and consistent time-based querying across the application.

```
1  {
2    "id": "manufacturing_exp",
3    "header": [
4      "value_6",
5      "value_7",
6      "value_4",
7      "value_5",
8      "value_2",
9      "value_3",
10     "value_1"
11  ],
12  "schema": "more",
13  "tableName": "manufacturing_exp",
14  "timeFormat": "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'",
15  "timeRange": {
16     "from": 1329946931000,
17     "to": 1330146930990,
18     "fromDate": "2012-02-22 21:42:11",
19     "toDate": "2012-02-25 05:15:30",
20     "intervalString": "[2012-02-22T21:42:11Z (1329946931000), 2012-02-25T05:15:30.990Z (1330146930990)]"
21  },
22  "samplingInterval": 10,
23  "measures": [
24    0,
25    1,
26    2,
27    3,
28    4,
29    5,
30    6
31  ]
32 }
```

Figure 3.6. The metadata response contains the names of the measures, the selected schema and the time range.



Handling the server's response. Once the metadata request is completed, the retrieved minimum and maximum timestamps are used to define the allowable bounds for the user's time range selection, ensuring that any queries remain within the dataset's temporal limits. Simultaneously, the list of available measures is used to populate a dropdown menu in the interface, allowing users to select one or more measures for visualization.

Whenever the user modifies the time range or selects a different set of measures, a new network request is sent to the backend API. This request includes all the relevant query parameters, such as the selected start and end timestamps ("from" and "to"), the chosen measures ("measures"), the pixel dimensions of the HTML element where the chart will be rendered ("width" and "height"), and any configuration parameters specific to the selected visualization method. These method-specific parameters are nested under the "methodConfig" field, which includes a "key" identifying the visualization method (e.g., "MinMaxCache-1742064733113") and a "params" object specifying the method's initialization settings such as `prefetchingFactor`, `aggFactor`, and `dataReductionRatio`, which differ from method to method.

Additionally, metadata about the selected schema and table ("schema": "more", "table": "manufacturing_exp") and any other optional query constraints (e.g., "accuracy": 0.95) are included to further refine the request. These inputs ensure that the backend can accurately prepare and return the data required to generate the appropriate visualization in the frontend.



```
1  {
2    "query": {
3      "methodConfig": {
4        "key": "MinMaxCache-1742064733113",
5        "params": {
6          "prefetchingFactor": 0,
7          "aggFactor": 4,
8          "dataReductionRatio": 6
9        }
10     },
11     "from": 1329946931000,
12     "to": 1329946991000,
13     "measures": [
14       4
15     ],
16     "width": 507,
17     "height": 260,
18     "schema": "more",
19     "table": "manufacturing_exp",
20     "params": {
21       "accuracy": 0.95
22     }
23   }
24 }
```

Figure 3.7. Request payload to fetch time series data. It contains one of the selected methods, the time range, the selected measures, the data source and the dimensions of the element that the data will be presented.

The API response returned by the backend contains the time series data required for rendering the visualization. Each data point is structured as a combination of a timestamp, a value, and a corresponding measure identifier, grouped under the "data" field. This format allows the frontend to associate multiple time series with their respective variables (e.g., different sensor readings or financial metrics).

In addition to the raw data points, the response includes a "timeRange" object that defines the start and end timestamps of the returned dataset, both in Unix epoch format and as ISO 8601 formatted strings (fromDate and toDate). This ensures both machine-readable precision and human-readable clarity. The "intervalString" further summarizes the queried time span in a single textual representation, aiding logging or debugging.

Performance metrics are also provided. The "queryTime" field records the duration in seconds it took for the backend to process the request and return the result. The "ioCount" field reflects the number of input/output operations involved in executing the query, giving insight into the data



access cost. Finally, the "metrics" object includes additional diagnostic information, such as visualization-specific error metrics (e.g., error rate and missing pixel data), which can be used to display the errors that a visualization method may have introduced to the generated visualization and its location in the rendered chart.

```

1  {
2    "message": "InfluxDB query executed successfully.",
3    "queryResults": {
4      "data": {
5        "4": [
6          ...
7          {
8            "timestamp": 1329946931000,
9            "value": 14436,
10           "measure": 4
11          },
12          ...
13        ]
14      },
15      "timeRange": {
16        "from": 1329946931000,
17        "to": 1329946990826,
18        "fromDate": "2012-02-22 21:42:11",
19        "toDate": "2012-02-22 21:43:10",
20        "intervalString": "[2012-02-22T21:42:11Z (1329946931000), 2012-02-22T21:43:10.826Z (1329946990826)]"
21      },
22      "queryTime": 0.094696547,
23      "ioCount": 5983,
24      "metrics": {
25        "error": "{4=ErrorResults [error=0.0, missingPixels=[[], ..., []]}"
26      }
27    }
28  }

```

Figure 3.8. Response payload containing the query results, the time range of the data, the execution time of the query and the number of data points.

3.1.2. Handling User Interaction

Time range can be adjusted either directly, by changing the time range in the Configuration Panel to directly request a specific time range, or indirectly by interacting with the chart itself, specifically through panning or zooming. These interactions allow users to explore different segments of the time series data in an intuitive and fluid manner. However, such interactivity introduces challenges on the system side. Because user input during panning or zooming can be rapid and continuous, the application may initiate multiple overlapping data requests to the backend as the visible time window updates with each interaction, which can overload the backend or the data store, or display incorrect time ranges due to out-of-order responses caused



by network delays. To address this, the application leverages the AbortController API, a built-in browser interface that allows ongoing HTTP requests to be programmatically cancelled. When a new interaction triggers a request, the application first terminates any previous requests that are still in progress. This ensures that the backend only processes the most recent user action, reducing computational overhead and improving overall responsiveness. This approach not only optimizes performance but also enhances user experience by preventing outdated data from being rendered after a newer request is already underway.

Handling data in JavaScript. All filter values are stored in the internal state of the Dashboard component. ReactJS provides the `useEffect` function that allows the developer to run a callback function when the component state is altered either partially or completely. This function is used to monitor the variable holding the time series data.

In its callback function, D3 is used to handle the rendering of the chart. Initially, it clears all data points, axes, and gridlines from the SVG. Then it calculates the x-axis scale using the timestamps included in the time series data and the available width of the chart. The y-axis scale is also calculated considering the maximum and minimum values that appear in the time series data, and the available height of the chart.

```
474 // Start from a pixel right of the axis
475 // End at the right edge
476 const x = d3
477   .scaleTime()
478   .domain([minTs, maxTs])
479   .range([margin.left + 1, Math.floor(width - margin.right)]); // Floor the width to avoid blurry lines
480
481 // Start from a pixel right of the axis
482 // End at the right edge
483 const minValue = d3.min(formattedData, (d: any) => d[1]);
484 const maxValue = d3.max(formattedData, (d: any) => d[1]);
485
486 // Start a pixel above the bottom axis
487 // End at the top edge
488 const y = d3
489   .scaleLinear()
490   .domain([minValue, maxValue])
491   .range([Math.floor(height - margin.bottom) - 1, margin.top]); // Floor the height to avoid blurry lines
```

Figure 3.9. Using the D3 scale, `domain()` and `range()` function, minimum and maximum timestamps and values are transformed into pixel coordinates



The scale calculation in TimeVizBench uses D3.js to transform time series data from its raw domain (timestamps and numeric values) into pixel coordinates that can be plotted on an SVG canvas. This transformation is essential for mapping abstract data into a form that can be rendered and understood visually.

As shown in the code snippet in Figure 3.9, the x-axis scale is defined using `d3.scaleTime()`, which is specifically suited for temporal data. Its domain is set to the minimum and maximum timestamps (`minTs`, `maxTs`) of the current time window. The corresponding range defines how these timestamps map to horizontal screen positions, starting from just inside the left margin (`margin.left + 1`) and ending at the right edge of the chart area (`Math.floor(width - margin.right)`). The use of `Math.floor` ensures that the scale aligns precisely with full-pixel boundaries, avoiding blurry line rendering caused by subpixel aliasing in browsers.

For the y-axis, the scale is computed using `d3.scaleLinear()`, which maps numeric values to vertical positions on the screen. Before defining the y-scale, the code calculates the minimum and maximum values (`minValue`, `maxValue`) from the current dataset using `d3.min()` and `d3.max()`, applied to the value component of each data point. The range of the y-axis is set from the bottom to the top of the chart (`height - margin.bottom` to `margin.top`), but notice that the range is reversed vertically (top comes after bottom) since in SVG coordinates, increasing y-values move downward. Here `Math.floor` is used as well, to align rendering to pixel edges and avoid visual artifacts.

Both the x and y scales are essential for generating the path of the time series line, which is computed by D3's `line()` generator. This generator takes in the scaled coordinates and applies the selected interpolation method, in this case `curveLinear`, which connects points using straight lines[6].

This setup ensures that the chart is rendered with high precision and visual clarity, while maintaining performance and consistency across devices with varying pixel densities.



```
546 // Add path
547 const line = d3
548   .line()
549   .x((d: any) => Math.floor(x(d[0])) + 1 / window.devicePixelRatio)
550   .y((d: any) => Math.floor(y(d[1])) + 1 / window.devicePixelRatio)
551   .curve(d3.curveLinear);
552
553 const path = chartPlane
554   .append('path')
555   .attr('class', 'path')
556   .datum(formattedData)
557   .attr('fill', 'none')
558   .attr('stroke', 'blue')
559   .attr('stroke-width', 1 / window.devicePixelRatio)
560   .style('shape-rendering', 'crispEdges')
561   .attr('d', line);
```

Figure 3.10. Using the D3 `line()` function, a curve is generated that connects the given pixels

Finally, a zoom handler is added using the `zoom()` D3 function that allows the developer to register handlers while transforming the target element and at the end of the transformation. Zooming effectively changes the time range displayed. While zooming, the time scale is transformed accordingly and the new path of the line is calculated and displayed using the new time scale and the same values, outputting a narrower or wider time range as a result of the users' input until the newly requested time range is rendered. On completing the action, the final time scale is calculated to get the new time domain which in turn triggers the function to fetch the time series data with the new time range.



```
564 const zoom = d3
565   .zoom()
566   .on('zoom', (event: any) => {
567     const newX = event.transform.rescaleX(x);
568     path.attr(
569       'd',
570       d3
571         .line()
572         .x((d: any) => Math.floor(newX(d[0])))
573         .y((d: any) => Math.floor(y(d[1])))
574         .curve(d3.curveLinear)
575     );
576     svg.selectAll('.point').remove();
577     svg.selectAll('.error-pixel').remove();
578   })
579   .on('end', (event: any) => {
580     const newX = event.transform.rescaleX(x);
581     let [start, end] = newX.domain().map((d: any) => dayjs(d.getTime()).toDate());
582     setFrom(start);
583     setTo(end);
584   });
585   svg.call(zoom);
```

Figure 3.11. Using the D3 zoom() function, a curve is generated that connects the given pixels

D3 retains a state regarding the zoom transformation that has taken place. In this case, the new time range is calculated in the backend, making the D3 zoom state obsolete. For that reason, in each render of the chart, the zoom transform is being reset.

3.1.3. Rendering Lines with Pixel Precision

Rendering data points and the lines that connect them can be challenging especially when the number of elements approaches or surpasses the available pixels of the output device that represent them. For applications where the data points are less than the available pixels, anti-aliasing can be used to make the resulting image smoother. In this case, each pixel needs to be controlled separately.

Off-the-shelf solutions' limitations. Popular off-the-shelf charting libraries like [Chart.js](#) and Plotly provide high-level abstractions for rendering visualizations with minimal setup. They provide a declarative API that uses JSON-like configuration objects that abstracts the rendering



logic, thus preventing low level access to pixel-level operations. Therefore, developers have limited or no control over how data is mapped onto individual pixels especially in density contexts where aliasing and overplotting may occur. In those cases, most libraries use layout strategies, interpolation, and anti-aliasing parameters that cannot be customized per pixel. The abstraction, as convenient as it is, prevents the developer from being able to control individual pixels. Because the goal was to produce a robust and reliable benchmarking solution, such a restriction demanded a lower-level solution that supports per-pixel rendering logic, a requirement that is met by [D3.js](#).

Data-to-pixel mapping. Once such low-level control is enabled, it becomes necessary to normalize coordinate input and account for device-specific pixel characteristics in order to ensure consistent and deterministic rendering behavior. To achieve this, integer pixel coordinates should be used and the CSS px unit used by SVG[7] should be converted to represent the physical pixels of the output device. The px unit is defined as the visual angle of one pixel of a device pixel density of 96 dpi from a distance of around 28 inches, where the device pixel is considered the area that contains the red, green and blue subpixel. A full HD screen (1920x1080 pixels) with a 24 inches diagonal has a pixel density of 91 pixels per inch meaning that 1 CSS pixel is rendered 1 physical pixel. For a screen with higher pixel density like the MacBook Retina display with resolution 2560 × 1600 pixels with a 13.3 inches diagonal, the pixel density is 227 pixels per inch so 1 CSS pixel will be rendered using a 2x2 pixels rectangle. This means that CSS px is not always 1:1 mapped to a physical pixel of the output device[8] and the CSS px size of a physical pixel will be calculated using $1/\text{devicePixelRatio}$. The device pixel ratio is supported by all known browsers. For a path in SVG to have a true single pixel width, the device pixel ratio should also be taken into consideration. This also ensures consistency across output devices with different resolutions and device pixel ratios.

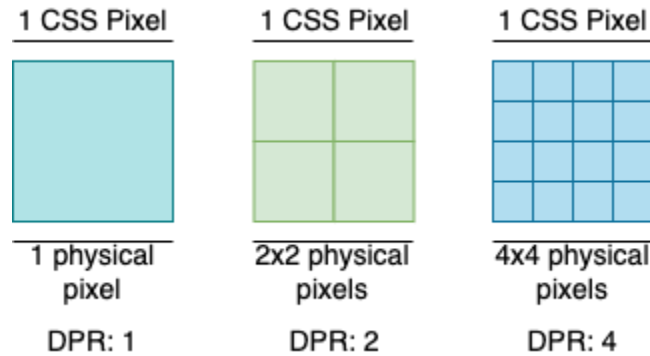


Figure 3.12. The number of physical pixels composing one CSS pixel depends on the Device Pixel Ratio (DPR)

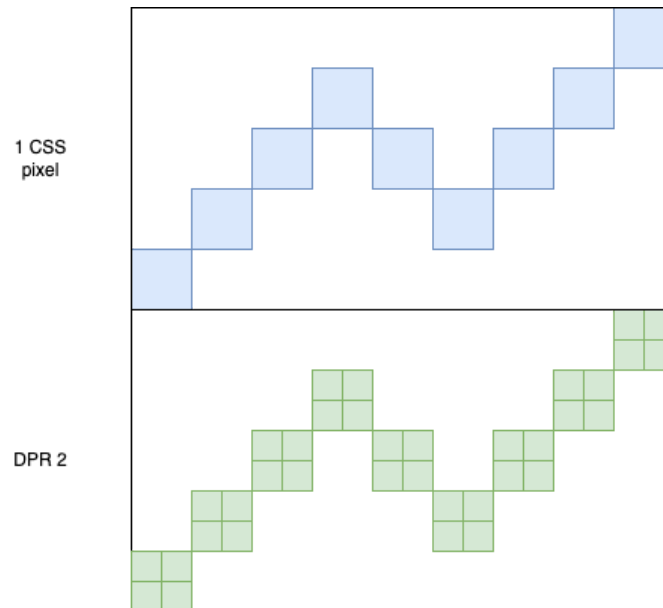


Figure 3.13. A line rendered in a 18x10 screen and device pixel ratio 2:1. The first line shows the line with 1px CSS width and the second line highlights the physical pixels used to compose it.

The path shown in Figure 3.11 is rendered using the D3 line object created using the time series data and passing to the d parameter of the SVG element which is used to draw a line in the browser. The line is instantiated using the scales created as shown in Figure 3.11. The scales generated by D3 have decimal numbers for the coordinates as output e.g. (2.25, 1.75) in CSS pixels. Rendering engines in browsers employ a technique called pixel snapping to handle decimal positioning of pixels in order to produce pixel perfect output avoiding blurriness or visible artifacts like the Moire effect[9]. This means that the coordinates (2.25, 1.75) will be

rendered in position (2, 2) by rounding each coordinate. To ensure the consistent rendering between different rendering engines, the coordinates of the data points are explicitly rounded down.

The additional transformation applied is to add 1 physical pixel to the horizontal and the vertical axis.

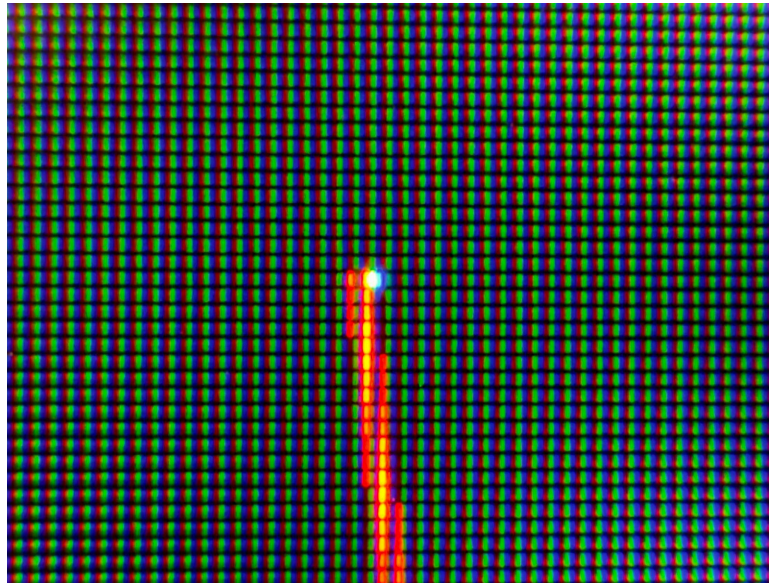


Figure 3.14. The line drawn by D3 starts one pixel to the left from the rendered data point

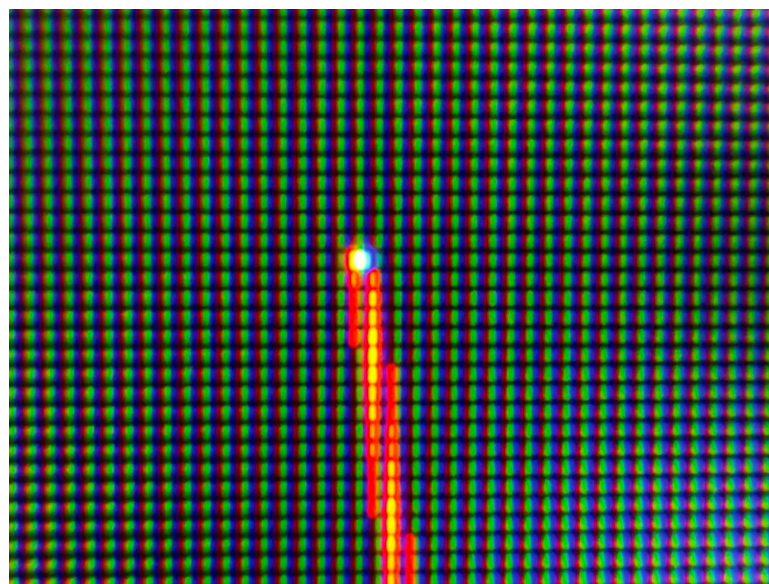


Figure 3.15. An offset is applied to the line to ensure that the line pass through the data point



Figures 3.14 and 3.15 are close up shots of a Full HD screen with 91 pixels per inch density. The white dot is a data point while the red line is generated using D3. The background is set to a dark gray, making the surrounding pixels visible enough to reveal the underlying red, green and blue subpixels. The white color was used for the data point to utilize the full luminance of all three subpixels, effectively rendering the whole physical and CSS, in this case, pixel. For the line, the red color was used to illuminate partially each pixel used to render it to give a clearer picture of the distance between pixels. As shown in Figure 3.14 and 3.15, the line that is rendered does not pass through the pixel despite being one of the pixels used to generate it. For that reason an offset is added ensuring that the line will pass through each data point, providing a consistent output. For example, using the statement from line 549 in Figure 3.11 to calculate the column of the pixel that will be used as part of a line along with its offset, with $x(d[0]) = 10.2$ and $devicePixelRatio = 2$ results in 10.5.

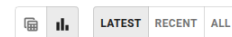
$$\text{Math.floor}(x(d[0])) + 1 / devicePixelRatio = 10.5$$

This means that the line will pass through the 11th column instead of the 10th without the offset.

3.1.4. Evaluating Visualization Method Performance

An additional tool that the frontend provides that assists with the evaluation of the selected visualization methods, are the evaluation metrics that are available in the Performance metrics panel. The first metric displayed is the total request time which is the time it took from the initial request up to the rendering of the line for each selected method. It is broken down in query time, which represents the total time taken by the data source to respond, in networking, and for the rendering to complete.

Query History



Operation at 6/29/2025, 7:14:31 PM



Figure 3.16. Bar charts displaying the time breakdown

The time breakdown is displayed using a stacked bar chart which provides an intuitive way to compare both the total time and the time of each individual step. A second bar chart is used to compare the number of data points used from each visualization method, enabling the user to compare the total amount of data retrieved. Both of the aforementioned metrics constitute the latency of each method.

The user can calculate the accuracy of the visualization by comparing the resulting line charts of each visualization method using the Structural Similarity index (SSIM). SSIM is a perception-based method that can be used to compare an image against a reference image by calculating their similarity. The similarity is expressed with a decimal value that belongs to the range $[-1, 1]$ where 1 indicates perfect similarity, 0 indicates no similarity and -1 indicates perfect anti-correlation[10].

3.2. User Interface

Upon logging in, users are presented with the Dashboard. From there they can interact with the filters that are available in the Configuration panel, and see the results in the Visualization panel, and the performance in the Performance Metrics panel.

The User Interface (UI) allows the users to select a time range, the measures available in the dataset, and visualization methods to compare their performance. Upon first load, the time range is set to the first minute of available data in a preselected dataset. Next, the user can select a

method instance specifying its initialization parameters, and finally the user selects the measures to be displayed. After fetching and rendering the charts, the user can evaluate each method's performance using the evaluation metrics in the Performance Metrics panel. Additionally, they can interact with the charts, zooming or panning to fetch additional data, and make a chart take over the full width of the screen displaying the chart in greater detail. Each query the user makes, is stored in the applications state and it is available for download in comma-separated values format.

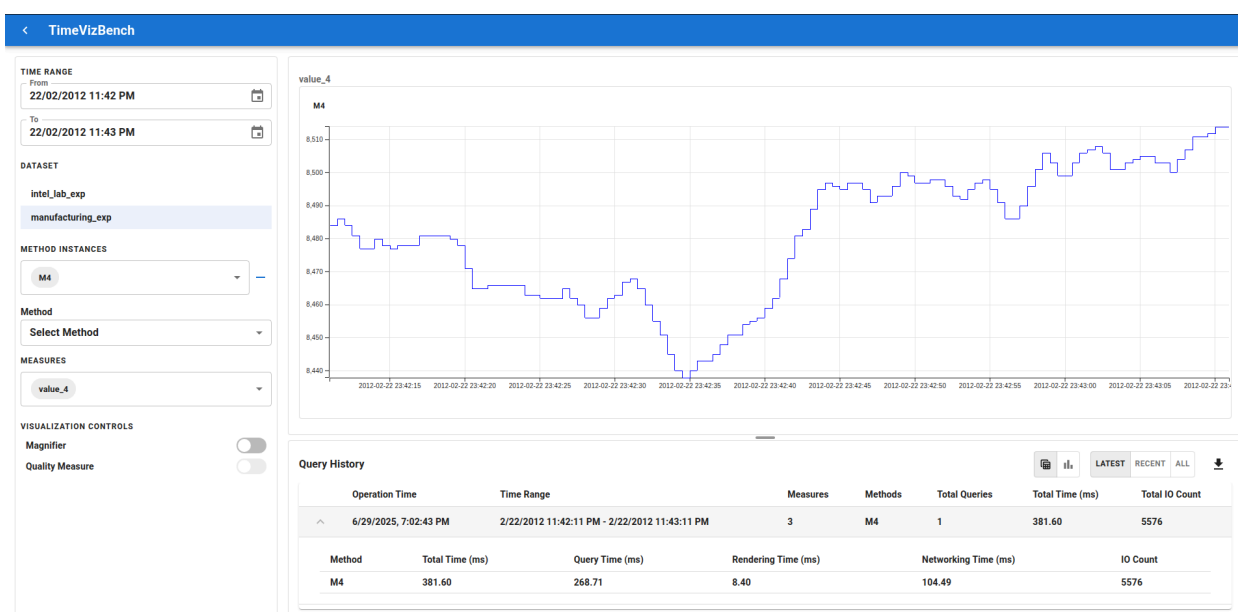


Figure 3.17: TimeVizBench Dashboard

3.2.1. UI Components

The user interface of TimeVizBench is organized into distinct visual components that support a structured and interactive workflow. Users select datasets, time ranges, and visualization methods in the Configuration panel, view the resulting charts in the Visualization panel, and analyze performance metrics in the Performance Metrics panel. The following subsections describe each of these UI elements in more detail, highlighting their roles in facilitating effective exploration and comparison of time series visualization techniques.

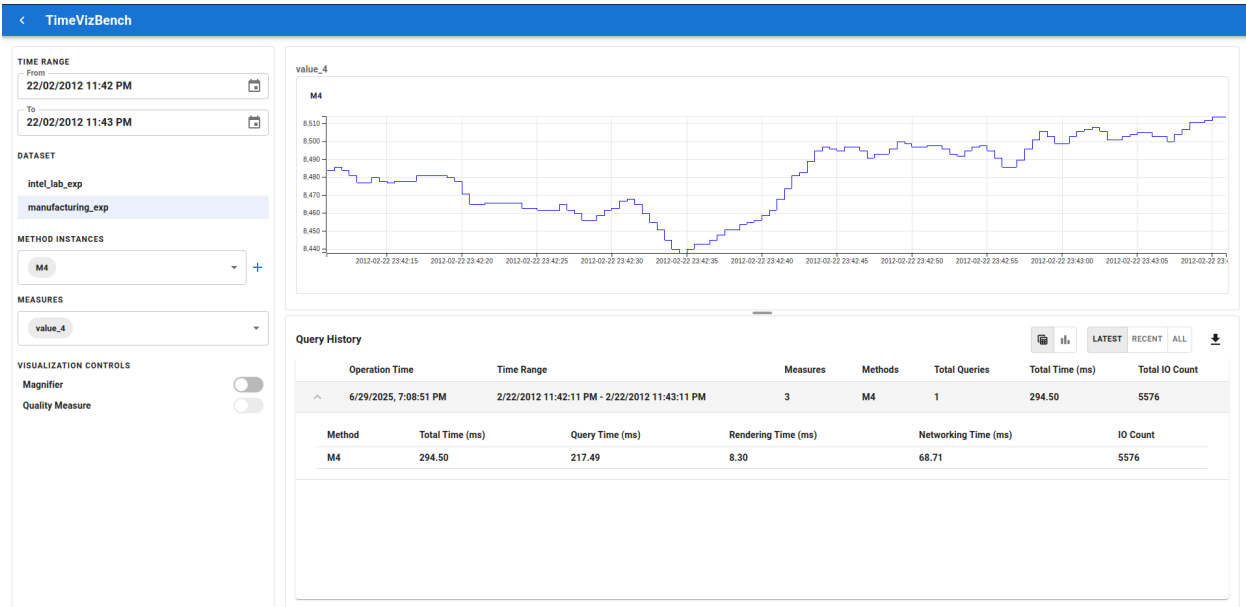


Figure 3.17. TimeVizBench Dashboard showing the query history.

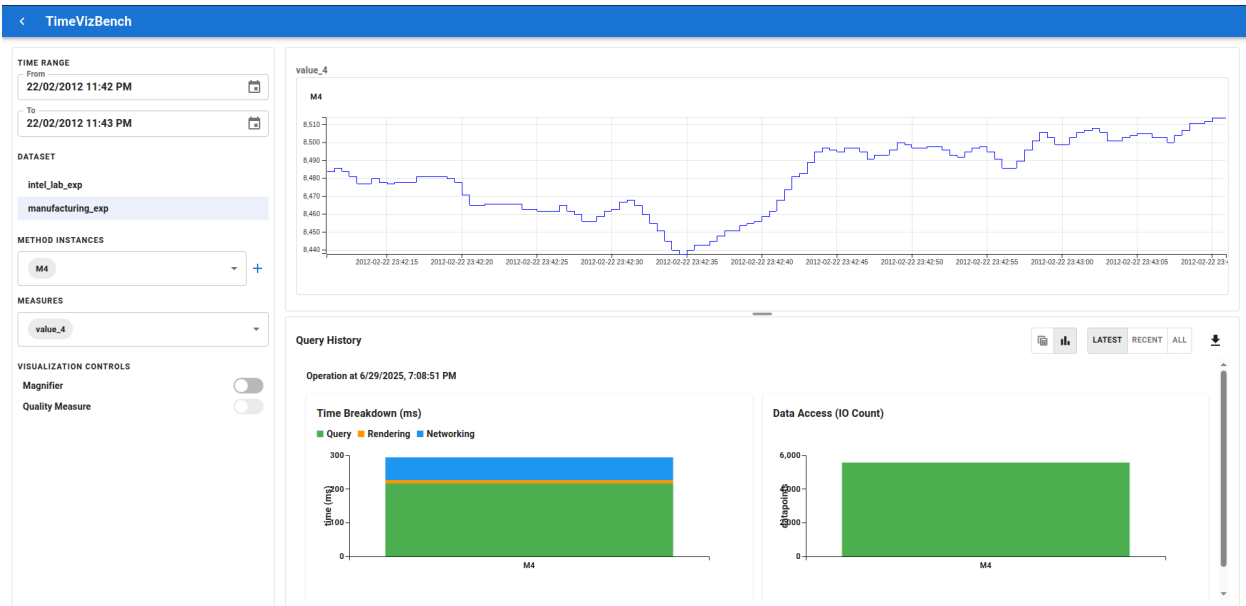


Figure 3.18. TimeVizBench Dashboard showing the time breakdown.

The SPA contains a root component shown in Figures 3.17, 3.18, that is mounted on a specific DOM element after the browser has loaded all the necessary scripts. The root component



consists of the Dashboard component that contains the Configuration panel, the Visualization panel and the Performance metrics panel.

The Configuration panel provides filters for the user to select datasets, the visualization method and its parameters for comparison, and the time range through interactive date and time pickers. In the Visualization panel D3 renders the charts using Scalable Vector Graphics (SVG). In the Dashboard component there are functions defined that are responsible for fetching metadata that describes the contents of a selected datasource, like the maximum and minimum timestamps of its data as well as the available measures that a user can select to display, and metadata about the implemented visualization methods.

3.2.1.1. Configuration Panel

The Configuration panel as shown in Fig. 3.19 contains the inputs from which the user can modify their query to evaluate the selected visualization methods. Firstly, it consists of two date and time pickers that are provided from the Material UI frontend library. Using those inputs, the user can define the time range of the time series data.



TIME RANGE

From
22/02/2012 11:42 PM

To
22/02/2012 11:43 PM

DATASET

intel_lab_exp

manufacturing_exp

METHOD INSTANCES

M4 MinMaxCache +

MEASURES

value_4

QUERY PARAMETERS

MinMaxCache

Accuracy
0.95

VISUALIZATION CONTROLS

Magnifier

Quality Measure

Figure 3.19: The control panel

Next, a list of datasets is available for the user to select. Each dataset contains a different set of measures to display. Next, a drop down list containing all the available visualization method instances that are implemented in the backend. The drop down list allows the user to select multiple methods to be displayed. Upon selecting a method, users are prompted to set any initialization parameters that any method may require.

Next, a drop down list is available providing all the measures contained in the selected dataset. The user can select multiple measures from a single dataset to be displayed using all the selected visualization methods. Finally, query parameters are method-specific parameters. The query parameters and the filters can be adjusted dynamically, allowing the user to interact with them, altering the resulting visualization. The method instantiation parameters set during method selection, remain the same for all queries that the method is selected.

Additionally, visualization controls offer two useful features that enable a more accurate assessment while exploring a dataset. The first is the magnifier allows the user to explore more easily the features of the rendered line chart by zooming in the area under the mouse pointer.

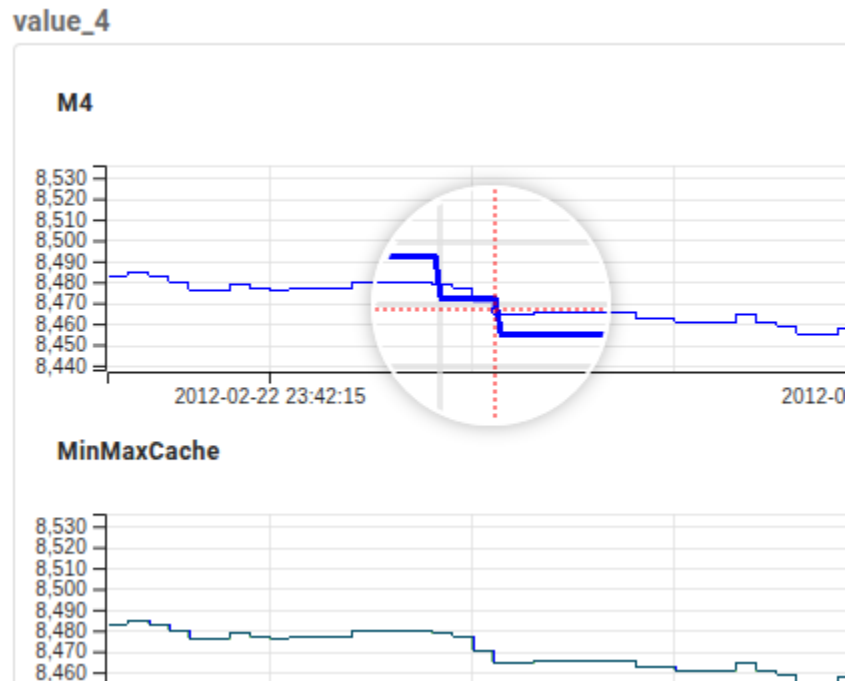


Figure 3.20: The magnifier enabled from the visualization controls

The second one is the quality measure. This option is enabled when there are more than 2 selected method instances and displays their similarity using SSIM inside the chart as shown in Fig 3.21.



Figure 3.21: SSIM score

3.2.1.2. Visualization Panel

The Visualization panel, as shown in Fig. 3.22, serves as the central display area for rendering time series line charts based on the user's selected measures and visualization methods. Each method generates its own chart instance, allowing side-by-side comparison of how different

techniques represent the same dataset. These visualizations are rendered using D3.js, which ensures scalable and interactive SVG-based graphics capable of handling large volumes of data.

Users can engage with the charts directly through intuitive interactions such as panning and zooming. Panning horizontally allows users to shift the visible time window backward or forward, effectively retrieving data from earlier or later time intervals. Zooming enables a more detailed view (zoom-in) or a broader overview (zoom-out) of the time series, dynamically adjusting the granularity of the displayed data.

This interactive design enhances the user's ability to explore temporal patterns, detect anomalies, and compare the performance and output quality of multiple visualization techniques under different zoom levels and time intervals.

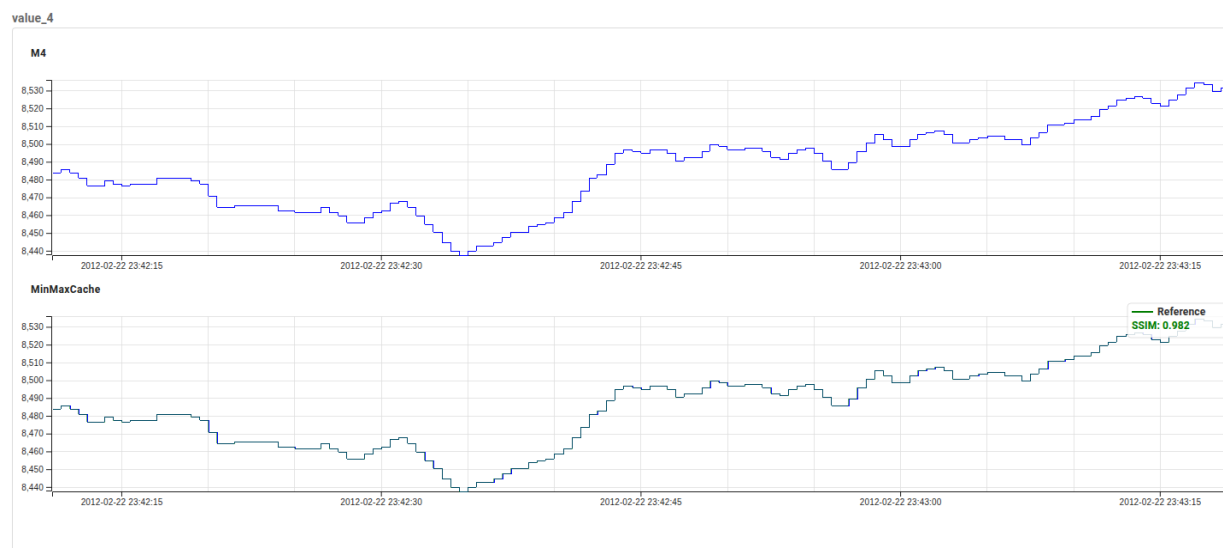


Figure 3.22: The visualization panel

3.2.1.3. Performance Metrics Panel

The Performance Metrics panel, as shown in Fig. 3.16, provides a detailed summary of the system's performance for the most recently executed query. In this panel, two tabs can be displayed, the query history and the visualized performance metrics of the operation.

Additionally, there is an option for both tabs, to display only the last operation, the five most recent ones and all the operations.

The query history is a list of all queries that have been performed in TimeVizBench as shown in Fig. 3.23. Each item contains the time at which the operation was performed, the parameters of the queries eg. the time range, selected measures, visualization methods etc. as well as the time breakdown as text.

Operation Time	Time Range	Measures	Methods	Total Queries	Total Time (ms)	Total IO Count																		
6/29/2025, 7:27:06 PM	2/22/2012 11:42:11 PM - 2/22/2012 11:43:19 PM	3	M4, MinMaxCache	2	442.50	5576																		
<table border="1"> <thead> <tr> <th>Method</th> <th>Total Time (ms)</th> <th>Query Time (ms)</th> <th>Rendering Time (ms)</th> <th>Networking Time (ms)</th> <th>IO Count</th> </tr> </thead> <tbody> <tr> <td>M4</td> <td>358.80</td> <td>234.19</td> <td>9.50</td> <td>115.11</td> <td>5576</td> </tr> <tr> <td>MinMaxCache</td> <td>83.70</td> <td>1.88</td> <td>9.40</td> <td>72.42</td> <td>0</td> </tr> </tbody> </table>							Method	Total Time (ms)	Query Time (ms)	Rendering Time (ms)	Networking Time (ms)	IO Count	M4	358.80	234.19	9.50	115.11	5576	MinMaxCache	83.70	1.88	9.40	72.42	0
Method	Total Time (ms)	Query Time (ms)	Rendering Time (ms)	Networking Time (ms)	IO Count																			
M4	358.80	234.19	9.50	115.11	5576																			
MinMaxCache	83.70	1.88	9.40	72.42	0																			
6/29/2025, 7:22:22 PM	2/22/2012 11:42:11 PM - 2/22/2012 11:43:19 PM	3	M4, MinMaxCache	2	406.30	5576																		
6/29/2025, 7:19:33 PM	2/22/2012 11:42:11 PM - 2/22/2012 11:43:19 PM	3	M4, MinMaxCache	2	596.20	12407																		

Figure 3.23: Query history

The visualized performance metrics tab shows two charts, one for time breakdown and one for the number of data points that were accessed for each one of the selected methods. Each chart captures a different aspect of performance, helping users better understand the trade-offs associated with the selected visualization method.

For the time breakdown, the total time required to complete a query is visualized using a stacked bar chart. This total time is composed of the entire lifecycle of the request, from the moment the network request is initiated by the frontend, through the query execution time on the data source, to the rendering time needed for the frontend to visualize the result. By breaking down the total latency into these three components, network latency, backend processing, and frontend rendering, allowing users to more effectively compare visualization methods, identifying which stage each method excels in or falls short. This insight is especially valuable when selecting a visualization technique based on specific performance priorities, such as low latency or minimal resource usage.

The number of accessed data points reflects the efficiency of the visualization method in terms of how much data is required to produce the rendered chart. For instance, a visualization method that leverages data reduction techniques, such as MinMaxCache, might retrieve significantly



fewer data points than one that renders the raw dataset like M4, resulting in faster response times and lower memory usage.

Together, these performance metrics offer both high-level and fine-grained insights, helping users understand the system's responsiveness and data efficiency. This information is particularly useful in contexts where interactivity and scalability are critical, such as monitoring applications, exploratory data analysis, or visualizing high-frequency time series data.

3.3. Adding a visualization method

TimeVizBench provides a modular and extensible backend architecture that enables developers to register custom visualization methods through an annotation-based auto-discovery mechanism. New methods can be integrated by implementing a predefined Java interface and annotating the class with metadata that describes its functionality and configurable parameters. This approach ensures seamless integration with the frontend, allowing newly defined methods to appear automatically in the user interface without requiring manual registration or hardcoded configurations.

To add a custom visualization method, a developer simply creates a new Java class within the *methods* package of the backend project. This class must implement the predefined *Method* interface. Additionally, the class must be annotated with *@VisualMethod*, a custom annotation that signals to the backend that the class should be included in the set of discoverable methods. The *@VisualMethod* annotation also allows developers to define a human-readable name and description for the method, which are displayed in the frontend interface for user selection.

If the custom method requires configuration parameters, such as control over data reduction levels, aggregation granularity, or visualization-specific behaviors, developers can define these using the *@Property* annotation. This annotation supports detailed parameter metadata, including the parameter name, type (e.g., integer, float), minimum and maximum values, default value, and step size for numeric inputs. These properties are automatically made available in the frontend configuration panel, where users can adjust them through dynamically generated form controls. Moreover, the *@Property* annotation also enables automatic validation, ensuring that user inputs



remain within the defined bounds and that only valid configurations are passed to the backend during runtime.

Developers can access the source code, API specification, and implementation guidelines through the official GitHub repository: <https://github.com/athenarc/TimeVizBench>.

4. Conclusions and Future Work

4.1. Conclusions

This thesis firstly presented the application of time series data along with the importance of visualization in enabling the user to extract useful information that would miss otherwise, like the trends that the data present or any patterns. Additionally, visualizing large-scale time series data presents its own challenges regarding retrieving a large volume of data for an accurate representation, which introduces latency, making it more difficult for the user to explore the data efficiently.

The state of the art visualization methods are also presented along with their trade-offs in terms of latency and accuracy, as well as their advantages over the traditional data reduction techniques that reduces the volume of data but lowers the accuracy. The visualization methods presented are M4, a visualization-aware method that aggregates data points according to their placement on the output device, reducing the total amount of data points needed to achieve the same representation effectively keeping accuracy at 100% compared to the raw dataset but reducing the latency.

OM3 is also presented as a multiresolution method that precomputes and stores min-max aggregates across multiple levels of granularity in a hierarchical structure. This approach enables progressive refinement of the visualization during user interactions such as zooming and panning, without requiring full recomputation for each view. By relying on precomputed aggregates, OM3 significantly reduces latency and ensures a smooth user experience. However, this comes at the cost of offline preprocessing and additional storage requirements. Despite these trade-offs, OM3 demonstrates how hierarchical approximation techniques can effectively support interactive exploration while maintaining visual fidelity.



Additionally, MinMaxCache is presented which provides a cache layer, holding data points from previous queries making consequent queries faster, a practice that follows closely the real-world application of data visualization where users make ad-hoc queries when exploring a dataset, while providing guarantees for the visualization errors that may introduce.

Because each visualization method has strengths and weaknesses, a standard way to evaluate them is necessary. State of the art benchmarks focus on evaluating the performance of database systems by generating data that emulate real-life scenarios in the context of each industry as well as a series of queries that either emulate the generation of reports with a static set of queries or emulate the usage of a user by executing a series of operations incrementally.

Visualization-specific benchmarks were also examined, which evaluate database systems in a more user-centric way by using real-world datasets, and utilizing a series of queries emulating the user interaction with a dashboard.

The methodology introduced in this thesis fills in the gap between evaluating the backend performance in terms of latency, and the accuracy of the graphical representation, providing an evaluation of the user's perspective. Providing an interactive way for exploring a selected dataset, users are able to define their own visualization methods and evaluate it against known methods in terms of latency broken down in query time, which gives insights for the backend performance that is correlated with the volume of data that is handling, and finally the networking, that also provides insights for how the volume of data affects the communication between the frontend and backend.

Additionally, the accuracy can also be quantified using SSIM, a state of the art image similarity measure that can provide insights regarding the similarity of the graphical representation of each method focusing on the features of each representation instead of the values of each pixel comprising it.

Moreover, this thesis presents the techniques employed to achieve a pixel perfect graphical representation of the time series data using D3.js, avoiding out of the box abstractions provided



by the browser and using low-level modification of the pixels used, ensuring the accuracy of the data points displayed as well as a consistent rendering among the selected visualization methods.

The techniques used were the rounding of pixel coordinates to an integer value, while taking into consideration the pixel density of the output device to avoid ambiguous values to be given to the browser and inconsistent images to be rendered due to pixel snapping thus lowering the credibility of the SSIM output. Moreover, the user journey followed in the front end application is described giving a clear image on how this tool can be used, and a description of how a user can define their own visualization method is provided.

4.2. Future Work

In future developments, TimeVizBench could be extended to include an interactive interface for defining custom queries on available datasets. Such a feature would allow users to construct parameterized queries involving operations such as filtering, grouping, and aggregation, enabling more flexible interaction with the underlying data. This would bring the tool closer to replicating the iterative and dynamic nature of real-world data exploration workflows. Furthermore, incorporating WebSocket-based communication would allow the system to support continuous data delivery, thereby enabling the evaluation of visualization methods under streaming conditions where data is incrementally updated over time.

5. Acknowledgements

I would like to express my sincerest gratitude to my thesis supervisor, Dr. Papastefanatos Georgios whose guidance proved invaluable. Additionally, I would like to thank Stamatopoulos Vassilis for his support and guidance on designing this solution as well as Dr. Maroulis Stavros for his guidance and the research material he provided. Finally, I would like to thank Dr Manolis Terrovitis and Prof. Damianos Chatziantoniou for serving in the master thesis committee and providing valuable comments.



References

- [1] Andrienko, G. L., Andrienko, N. V., Drucker, S. M., Fekete, J.-D., Fisher, D., Idreos, S., Kraska, T., Li, G., Ma, K.-L., Mackinlay, J. D., Oulasvirta, A., Schreck, T., Schumann, H., Stonebraker, M., Auber, D., Bikakis, N., Chrysanthis, P. K., Papastefanatos, G., & Sharaf, M. A. (2020). Big data visualization and analytics: Future research challenges and emerging applications. In Proceedings of the Workshops of the EDBT/ICDT 2020 Joint Conference (Vol. 2578). CEUR-WS.org. <https://ceur-ws.org/Vol-2578/BigVis1.pdf>
- [2] Jugel, Uwe, Zbigniew Jerzak, Gregor Hackenbroich, and Volker Markl. 2014. "M4: A Visualization-Oriented Time Series Data Aggregation." *Proceedings of the VLDB Endowment* 7 (10): 797 - 808. 10.14778/2732951.2732953.
- [3] Transaction Processing Performance Council. 2021. "TPC Benchmark™ H Standard Specification, Revision 2.17.1." https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf.
- [4] Transaction Processing Performance Council. 2021. "TPC Benchmark™ DS Standard Specification, Revision 2.6.0." https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.6.0.pdf.
- [5] Sanchez, Jimi. 2016. "A Review of Star Schema Benchmark." *arXiv preprint arXiv:1606.00295*. <https://doi.org/10.48550/arXiv.1606.00295>.
- [6] "Curves." n.d. D3.js. Accessed March 9, 2025. <https://d3js.org/d3-shape/curve#curveLinear>
- [7] W3C. (2018, October 4). *Scalable Vector Graphics (SVG) 2 – Coordinates and Units*. W3C. Retrieved February 22, 2025, from <https://www.w3.org/TR/SVG2/coords.html#Units>
- [8] W3C. (2023, June 19). *CSS Values and Units Module Level 3*. W3C. Retrieved February 22, 2025, from <https://www.w3.org/TR/css3-values/#reference-pixel>



- [9] “Device Pixel Content Box.” n.d. web.dev. Accessed March 10, 2025. Last updated July 07, 2020, from <https://web.dev/articles/device-pixel-content-box>
- [10] Wang, Zhou, 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): 600–612. 10.1109/TIP.2003.819861.
- [11] Khelifati, Abdelouahab, Mourad Khayati, Anton Dignös, Djellel Difallah, and Philippe Cudré-Mauroux. 2023. “TSM-Bench: Benchmarking Time Series Database Systems for Monitoring Applications.” *Proceedings of the VLDB Endowment (PVLDB)* 16 (11): 3363 - 3376. 10.14778/3611479.3611532.
- [12] Philipp Eichmann, Emanuel Zraggen, Carsten Binnig, and Tim Kraska. 2020. “IDEBench: A Benchmark for Interactive Data Exploration.” In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1555–1569. <https://doi.org/10.1145/3318464.3380574>
- [13] Joanna Purich, Anthony Wise, and Leilani Battle. 2025. An Adaptive Benchmark for Modeling User Exploration of Large Datasets. *Proc. ACM Manag. Data* 3, 1, Article 8 (February 2025), 24 pages. <https://doi.org/10.1145/3709658>
- [14] 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 (April 2024), 2091–2103. doi:10. 14778/3659437.3659460
- [15] 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.
- [16] Stamatopoulos, V., Maroulis, S., Pantoleon, C., Papastefanatos, G., & Vassiliadis, P. (2025). TimeVizBench: An interactive platform for evaluating techniques for efficient large time



series visualization. In Proceedings of the 29th European Conference on Advances in Databases and Information Systems (ADBIS 2025), Tampere, Finland, September 23–26, 2025.

- [17] Stamatopoulos, V., Maroulis, S., Kozanis, K., Psarros, I., Papastefanatos, G., Giannopoulos, G., & Terrovitis, M. (2023). A tool for visual exploration and analysis of solar photovoltaic module data. In Proceedings of the Workshops of the EDBT/ICDT 2023 Joint Conference (Vol. 3379). CEUR-WS.org.
https://ceur-ws.org/Vol-3379/BigVis2023_704.pdf