Evaluation of Databases for Digital Twins and Industrial Internet of Things: A Comparative Analysis

Bauyrzhan Amirkhanov [1], Azim Aidynuly [1,*], Murat Kunelbayev [1,2], Gulshat Amirkhanova [1], Timur Ishmurzin [1], and Dinara Zhaisanova [1]

[1] Department of Artificial Intelligence and Big Data, Faculty of Information Technology, Al-Farabi Kazakh National University, Almaty, Kazakhstan
[2] Laboratory of Artificial Intelligence and Robotics, Institute of Information and Computational Technologies, Ministry of Education and Science of Kazakhstan, Almaty, Kazakhstan
Email: amirkhanov.b@gmail.com (B.A.); azimaidynuly1@gmail.com (A.A.); murat7508@yandex.kz (M.K.); gulshat.aa@gmail.com (G.A.); timon.ishmurzin@gmail.com (T.I.); zhaisanova15@gmail.com (D.Z.)
*Corresponding author

***Abstract*—The adoption of Digital Twins (DT) and Industrial Internet of Things (IIoT) systems necessitates efficient database solutions for real-time data ingestion and analytics. This study evaluates the performance of time-series databases, Influx Database (InfluxDB) and Timescale Database (TimescaleDB), alongside Not only Structured Query Language (NoSQL) database Mongo Database (MongoDB). Through comprehensive benchmarking, including write throughput and query latency under simulated IIoT workloads, the study identifies trade-offs between write-intensive and read-intensive operations. The results highlight the suitability of InfluxDB for high-frequency data ingestion and TimescaleDB for complex analytical queries. The findings provide actionable recommendations for database selection in digital twin architectures, offering insights for practitioners in industrial applications. Key features and differences, such as data write/read speed and scalability, are analysed. Special attention was given to load testing using Go language, which allowed running parallel threads and achieving write speeds up to 300,000 records per second in InfluxDB. TimescaleDB showed stable performance when executing complex SQL queries, providing 40 ms per query when sampling 50,000 and 250,000 rows. Examples of using time series databases for storing and processing real-time data from IoT sensors are considered. A brief analysis of the OpenTwins architecture, its databases, and internal components related to database operations has been conducted. It is concluded that the choice of technology should be based on specific requirements for data processing speed, analytics, and long-term storage.**

***Keywords*—databases, digital twin, industrial internet of things, internet of things, time-series databases**

I. Introduction

The rapid development of the Industrial Internet of Things (IIoT) and the increasing adoption of digital twins are fundamentally changing the approach to data collection, processing and analysis in industry. Digital twins are virtual models that reflect the state and behaviour of physical objects in real time, enabling enterprises to improve operational efficiency, provide predictive maintenance, and enhance decision making. A key technology enabled by the IIoT is the Digital Twin (DT), a virtual model synchronized with a physical object or system through a continuous flow of real-time data [1, 2]. The industrial applications of such models are vast and continue to expand [3]. In the context of this study, a Digital Twin is defined not by its applications, but by its core data-centric architecture: a high-fidelity virtual representation that is dynamically updated with data from its physical counterpart, enabling analysis, simulation, and prediction, the novelty of this study lies in contextualizing these performance metrics specifically for the demanding data persistence layer of an IIoT-enabled Digital Twin. To highlight the broader implications, recent studies have explored the integration of digital twins into the framework of the IIoT, emphasizing the importance of scalable and adaptive architectures to address emerging challenges [4, 5]. The challenge of securely integrating IoT systems with cloud platforms has been explored, offering solutions for ensuring data integrity in real-time applications [6]. Digital twins have revolutionized smart manufacturing by providing real-time insights and predictions for factory operations [7]. The core challenge in digital twin and IIoT architectures is selecting a database system that balances high data ingestion rates, which are critical for real-time monitoring of thousands of sensors, with complex analytics necessary for predictive maintenance and long-term trend analysis, as well as scalability and reliability to ensure seamless operation under varying workloads.

Recent reviews have underscored the gaps in current implementations, noting the challenges in achieving seamless integration of IoT systems and digital twins,

Manuscript received April 18, 2025; revised July 29, 2025; accepted August 12, 2025; published January 15, 2026.

particularly in applications demanding high-speed data ingestion and analytics, a challenge highlighted in foundational studies on the future of Digital Twins [8–10]. Traditional relational databases, such as PostgreSQL, struggle to handle the high-frequency data ingestion demands of IIoT. Time-series databases, such as Influx Database (InfluxDB), excel in write speed but face limitations in performing complex analytical querying. Not only Structured Query Language (NoSQL) solutions, such as Mongo Database (MongoDB), provide schema flexibility, making them suitable for dynamic entity representation in digital twins, but lack optimization for time-series data. These constraints hinder the effective deployment of digital twin systems in real-world industrial settings, where both speed and analytical depth are critical.

At the core of digital twin and IIoT architectures are data storage and processing systems that can cope with streams of information from various sources: sensors, machines, and industrial devices. One of the key challenges is the selection of suitable database technologies. The variety of formats, volumes and velocity of incoming data requires the use of optimal solutions on which the scalability, performance and reliability of the system depend [11–13]. Moreover, studies have highlighted the critical role of hybrid architectures in combining the strengths of different database systems to overcome performance bottlenecks and meet the demands of complex digital twin ecosystems [13, 14].

A special role is played by time series databases (e.g., InfluxDB, Timescale Database (TimescaleDB)) that process time series data in real time, as well as NoSQL solutions (e.g., MongoDB) that allow flexible handling of unstructured data. A time series database, based on the name, is a database system that is specifically designed to handle time-series related information. Time series databases differ from the usual relational (PostgreSQL) and NoSQL (MongoDB) databases.

Time series databases such as InfluxDB and TimescaleDB are optimised to efficiently process huge volumes of time-stamped data. They prioritise time-based indexing where each data point has a timestamp attached to it. This allows time-series databases to work well with real-time analytics, IoT sensor data, and monitoring applications. Their ability to aggregate and downsample data over time makes them a critical component in scenarios involving continuous data ingestion [15].

Time series databases such as InfluxDB and TimescaleDB are optimised to handle huge continuous streams of data, making them ideal for scenarios such as real-time monitoring in IoT systems, tracking financial markets, and scientific experiments with continuous sensor readings. Studies have shown that these databases outperform traditional databases in both write throughput and query performance for time-based data. For example, Barez [16] conducted comprehensive comparative tests of time series databases with traditional relational systems and showed that InfluxDB is much more efficient in handling high write and read loads when the data is time-stamped.

InfluxDB is an open source time series database system optimised for timestamping, making it ideal for IoT applications, real-time data monitoring and analysis. It efficiently handles high write and query workloads through an architecture based on time sharding and the use of Time-Structured Merge tree (TSM). The new version of InfluxDB 3.0 (IOx) adds improved data processing capabilities through Apache Arrow technology that enables SQL support and provides scalability, efficient data compression and parallel query execution [17, 18].

InfluxDB is widely recognized for its optimization in high-speed data ingestion through its Write-Ahead Log (WAL) and time-sharded architecture [17, 18]. However, its SQL-like query language limits its ability to perform complex data analysis. In contrast, TimescaleDB, built as an extension of PostgreSQL, offers strong SQL support and excels in analytical queries, yet its write performance can become a bottleneck in high-frequency IIoT environments. MongoDB supports flexible schemas, making it suitable for representing dynamic entities in digital twins, but its lack of time-series optimization diminishes its effectiveness in handling large-scale temporal data. While each of these databases offers unique advantages, no single solution effectively addresses the dual demands of high-speed data ingestion and robust analytics.

InfluxDB categorises data into dimensions, tags, and fields. Dimensions are used to group data logically, tags index data for fast retrieval, and fields store the actual data values. All data is linked to timestamps for high efficiency when processing time series. Data is written via Write-Ahead Log (WAL) and then converted into a TSM structure for long-term storage. This allows for high data write and read speeds. InfluxDB 3.0 introduces the new architecture shown in Fig. 1 with support for columnar data storage and SQL queries. In addition to this, the database utilizes Iron Oxide (IOx) for improved data processing with high cardinality and efficient storage of large amounts of data [18].

TimescaleDB is an open-source extension for PostgreSQL specifically designed to efficiently work with time series data using PostgreSQL's powerful query processing capabilities. Unlike many standalone time series databases, TimescaleDB integrates with PostgreSQL, offering a familiar SQL interface and optimizing performance and scalability for time series workloads. TimescaleDB also supports seamless scaling by distributing data across multiple nodes while maintaining high availability and performance. Its integration with PostgreSQL ensures compatibility with various indexing methods, triggers, and stored procedures, making it a versatile solution for developers and data analysts.

TimescaleDB presents hypertables, which are virtual tables spanning multiple chunks of underlying data. Each chunk corresponds to a specific time range, allowing the data to be automatically partitioned. This design provides smooth scaling and improves query performance by focusing queries on relevant chunks rather than scanning the entire dataset. The database also supports automatic

partitioning in time and space, meaning that data is automatically distributed across multiple nodes, providing horizontal scaling [19, 20]. TimescaleDB has built-in support for data retention policies, allowing users to automatically delete old data after a specified period of time. This simplifies the lifecycle management of time series data without manual intervention, reduces storage costs and improves system performance [21].

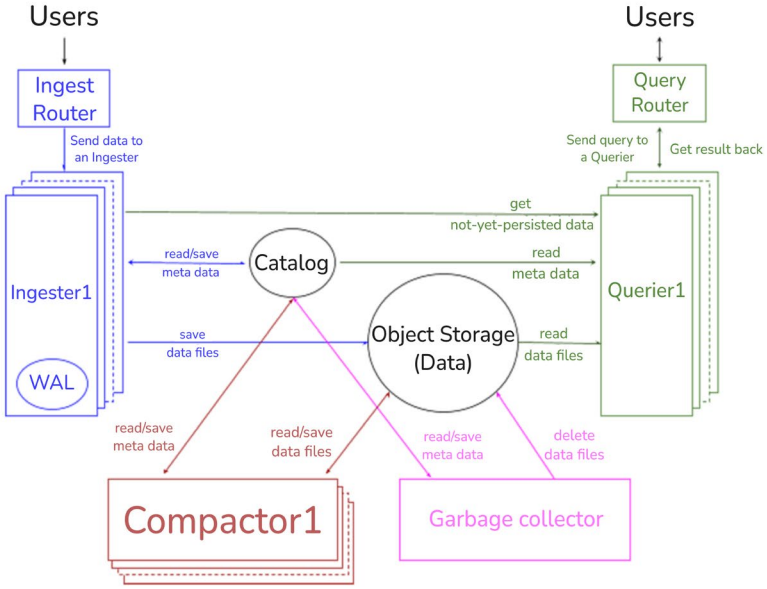

Fig. 1. InfluxDB 3.0 architecture reproduced from [18].

To cope with the large volume of repetitive queries on time series data, TimescaleDB implements continuous aggregates. These materialized views are automatically updated as new data becomes available, which significantly reduces the query execution time from minutes to milliseconds in many cases. One of the key benefits of TimescaleDB is its compression mechanism, which can reduce the amount of data stored by up to 90%. It uses native columnar compression applied to each chunk, which allows old data to be stored in a column-oriented format for efficient aggregation and fresh data to be stored in a row-oriented format for fast access [22].

MongoDB is widely used in digital twin systems to store entities, primarily due to its flexible schema-free document model, which is ideal for managing dynamic and complex data structures commonly found in digital twins. In the context of digital twin architectures, MongoDB does an excellent job of handling hierarchical and relational data models that represent entities and their relationships in the digital twin ecosystem. For example, digital vehicle twin projects use MongoDB to store digital representations of vehicles and synchronize them with real-time data from the physical world. In digital twins for IoT and large industrial systems, MongoDB's distributed architecture allows handling huge data sets and entity representations on multiple nodes, providing high availability and low latency, while its horizontal scalability supports the growing needs of interconnected digital twin environments [23, 24].

The architecture of digital twins often involves real-time monitoring and synchronisation of the physical and digital systems, as demonstrated in recent flexible manufacturing systems [25]. Recent studies have shown the potential of deep reinforcement learning to optimize manufacturing processes using digital twins, providing automated control systems that reduce costs and improve efficiency [26]. The integration of Cyber-Physical Systems (CPS) with digital twins plays a vital role in ensuring accurate control and simulation of production processes [27]. The OpenTwins architecture in Fig. 2 shows InfluxDB, and MongoDB are an integral part of its functionality, but serve different purposes [28].

InfluxDB is used as a time series database in OpenTwins. Its role is to manage sensor data and other time-sensitive information. In particular, it stores real-time data generated by IoT devices connected to the digital twin. In this architecture, sensor data is fed through tools such as Telegraf, which connects to Apache Kafka to stream data. For example, in the OpenTwins use case in the petrochemical industry, InfluxDB stores sensor data on the freezing temperature of lubricants. This data is then used in predictive models (via Kafka-ML) and visualized in Grafana, allowing plant operators to monitor and predict freezing temperatures in real time.

MongoDB is used in OpenTwins to manage entities rather than time series of data. It handles non-relational structured data representing digital twin entities. The flexibility of the MongoDB schema makes it an excellent choice for storing hierarchical structures defining digital twins and their subcomponents. This is particularly useful for managing complex relationships between different parts of a digital twin, such as machines, sensors, or even subsystems, where each entity may have dynamic attributes and properties that change over time. In the OpenTwins architecture, MongoDB is responsible for storing persistent data related to user configurations, entity states, and twin compositions, allowing easy updating and retrieval as the digital twin evolves.

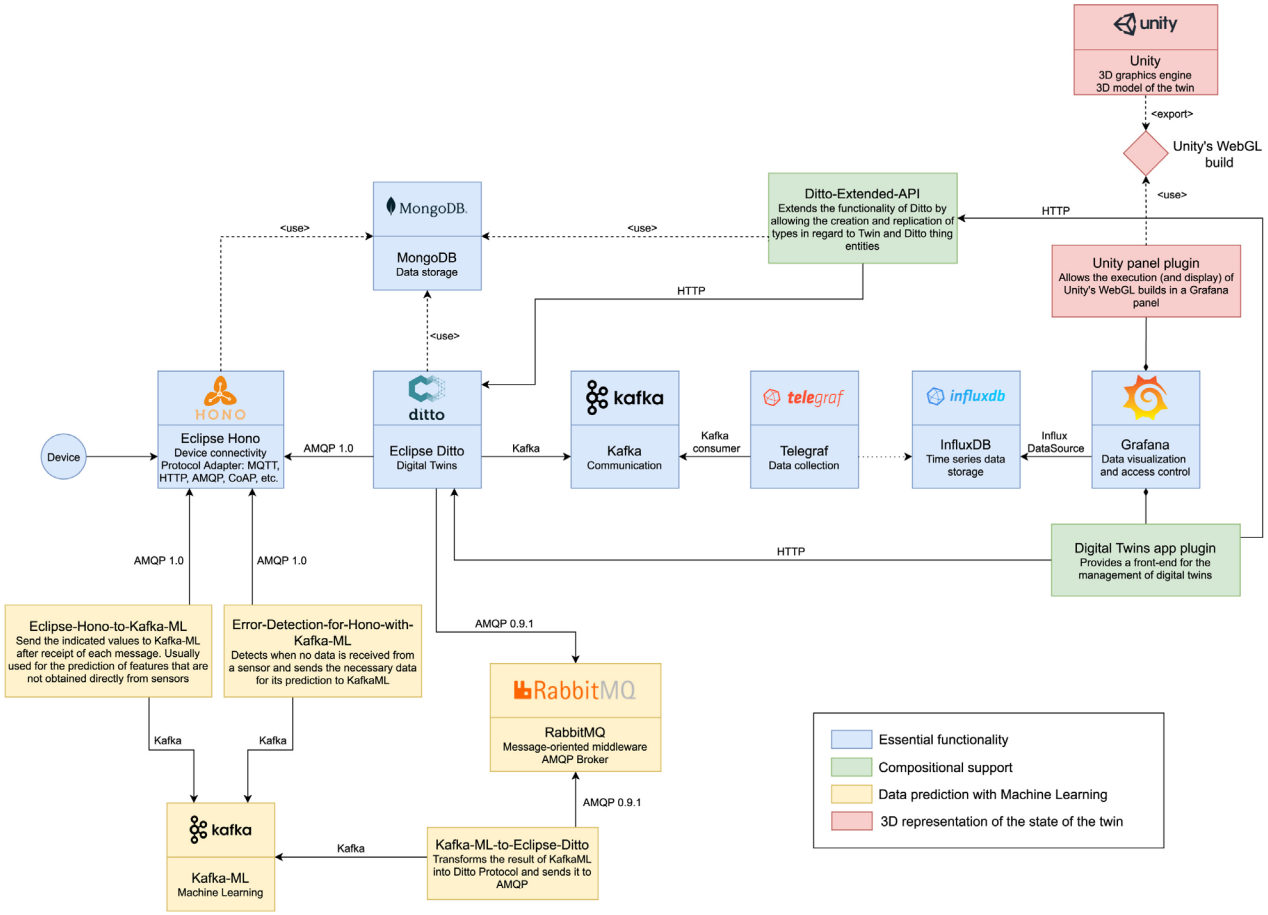

Fig. 2. OpenTwins architecture reproduced from [28].

This division of responsibilities between InfluxDB for time series data and MongoDB for entity management ensures that both real-time data streams and complex dynamic data structures can be efficiently handled in the OpenTwins framework. The concept of Cyber-Physical Production Systems (CPPS) is closely linked to the development of digital twins in Industry 4.0, enabling more adaptive and responsive systems [29]. The integration of digital twins with big data analytics is critical for smart manufacturing, offering 360-degree visibility into processes and operations [30]. The convergence of digital twins and multimedia technologies is shaping new applications, particularly in immersive environments [31].

The objective of this study is to provide a quantitative performance evaluation of InfluxDB and TimescaleDB under simulated IIoT workloads, focusing on write throughput and query latency. The research aims to compare these databases to identify trade-offs between write-intensive and read-intensive operations, as well as to provide actionable recommendations for selecting or combining database technologies based on specific use cases in digital twin and IIoT systems. By addressing these objectives, this research seeks to fill a critical gap in the literature and guide practitioners in choosing the right database architecture for their digital twin implementations. While benchmarks of individual database systems exist, the novelty of this study lies in contextualizing these performance metrics specifically for the demanding data persistence layer of an IIoT-enabled Digital Twin. This research provides a direct, quantitative comparison to validate the hybrid architectural model that is often discussed theoretically but lacks specific, comparative performance data to support its implementation.

The findings of this study will help industries optimize their database strategies for digital twins and IIoT systems, highlighting the strengths and limitations of existing technologies. Additionally, this research will offer insights into potential hybrid architectures that combine the advantages of multiple database systems, enabling more efficient and scalable solutions for the challenges faced by modern industrial environments. By evaluating real-world use cases and performance benchmarks, this study will provide practical recommendations for implementing robust and future-proof data management strategies.

II. Methodology

To evaluate the performance of InfluxDB and TimescaleDB, a benchmarking system was developed to simulate the data-handling requirements of the persistence layer of a Digital Twin, where high-frequency data from IIoT sensors must be ingested and made available for analytics. The complete source code for the benchmarking tool is available as supplementary material to ensure reproducibility [32].

A. Experimental Setup

In this study, a software system consisting of several key components was used to benchmark the InfluxDB and TimescaleDB databases. The main goal was to evaluate the performance of each database during data write and read operations. For this purpose, the Go programming language was chosen due to its established reputation for high performance and native support for concurrency. These features were critical for developing a high-throughput benchmarking tool capable of managing large data volumes and processing database queries in parallel. Furthermore, Go's standard library provides high-resolution timing functions, which offered the necessary accuracy for measuring execution times at the millisecond-level granularity required by this study. This capability, combined with simplified interaction with database Application Programming Interfaces (APIs), ensured the reliable collection of performance data [33, 34]. In addition, the system was deployed using Docker containers, which made it possible to quickly and easily isolate work environments for each database. This made it possible to ensure the same conditions for both tested systems and to avoid the influence of external factors, such as differences in hardware configuration. Docker containers made it easy to set up and manage databases, ensuring reproducibility of experiments and accuracy of the results obtained. Each database, both InfluxDB and TimescaleDB, was run in a separate container, which ensured their independent operation and excluded the influence of one database on another. This architecture made it possible to conduct parallel benchmarks, evaluate their performance under real load conditions, and analyse key metrics such as write speed and response time when reading data.

Fig. 3 shows the architecture of the benchmark. The overall architecture of the benchmark, shown in Fig. 3, is an adaptation of a benchmark for time-series databases in Scientific Experiments and Industrial Internet of Things (SciTS) framework proposed by Mostafa *et al*. [35]. The processes in the architecture begin with the configurator, which sets all the necessary parameters for correct connection to databases and additional parameters for adjusting the frequency of requests. After that, the module is configured to connect several clients to the database in parallel for multithreaded load and database testing, a data generation module is enabled inside the module, namely time series, after which the data is transferred to the benchmark logic processing module, where the processes of creating and preparing database queries take place, after that the queries are sent to the abstraction layer work on the database and further access to the database itself. The internal monitoring module collects performance information and outputs the result in a CSV (Comma-Separated Values) file format.

Docker containers have been selected for convenient and fast database deployment. In Fig. 4, both databases were deployed in isolated environments using Docker containers to ensure consistent and reproducible testing conditions. InfluxDB was configured to optimize write performance and long-term data storage. The Write-Ahead Log (WAL) was enabled to allow for high-speed data ingestion by buffering writes before committing them to disk. The Time-Structured Merge tree (TSM) storage system was configured to manage time-series data efficiently, with optimizations for compression and storage. A default retention policy was applied, allowing data to persist without additional truncation during the tests. TimescaleDB was configured to leverage its strengths in querying and data management. Hypertables were used to partition data into chunks based on time intervals, enhancing query performance and scalability. Continuous aggregates were enabled to precompute and store results for common queries, reducing overhead during benchmarking. The database inherited PostgreSQL's default settings for query execution and indexing, with hypertable-specific optimizations applied. Both databases were allocated identical resources within their Docker containers, including equal CPU (Central Processing Unit) shares to ensure comparable processing power, the same amount of Random Access Memory (RAM) to handle data caching and processing, and identical storage configurations to standardize read and write performance. This standardized setup minimized variability and ensured the results reflected the inherent performance characteristics of the databases under similar workloads.

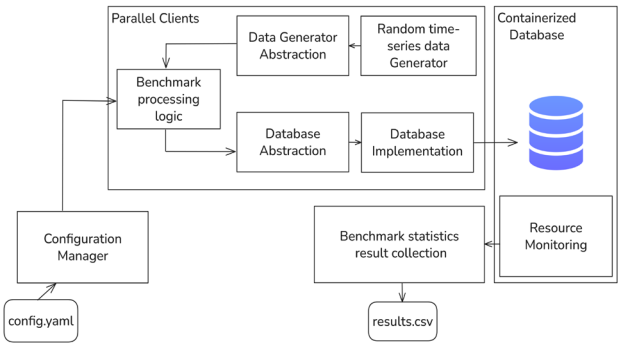

Fig. 3. The architecture of the benchmark.

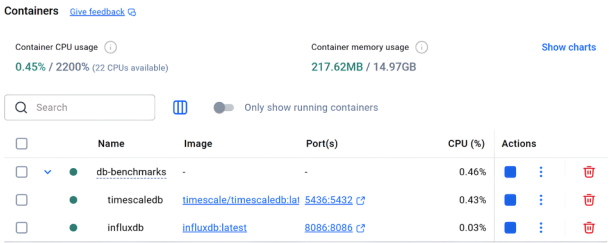

Fig. 4. Databases in docker containers.

The Go programming language was used to write a program that implements two benchmarks: writing and reading data into each database. Go was chosen due to its high performance and built-in support for competitive tasks. The benchmark program was written using third-party libraries and the Go standard library for working with database queries, as well as libraries for interacting with the InfluxDB and TimescaleDB APIs.

B. Benchmark Implementation

To evaluate the performance of InfluxDB and TimescaleDB, a benchmarking system was developed to simulate the data-handling requirements of the persistence layer of a Digital Twin, where high-frequency data from IIoT sensors must be ingested and made available for analytics.

The Go program implements benchmarks for writing and reading data in each database. The logic for these benchmarks was designed to test write speeds with a continuous stream of data and read speeds with analytical queries on datasets of 50,000 and 250,000 rows. During testing, each database processed requests for data sampling, and the response time was recorded for comparative analysis.

Benchmark Timescalewrite and Benchmark Timescaleread are modules responsible for testing the speed of writing data to the database, the speed of reading data, with different amounts of data: 50,000 and 250,000 rows. During testing, rows containing three parameters were recorded in each database: the sensor ID, the measured temperature and the timestamp. Each database processed requests for data sampling, and the response time was recorded and used for comparative analysis.

C. Data Generation

Generated data simulating the behavior of temperature sensors were used to conduct the experiment. Three parameters were stored as input data in the databases. Sensor ID: the unique identifier of each sensor from which the data comes. Temperature: the temperature values that were generated for each sensor. Timestamp: The time at which the temperature measurement was recorded.

$$Sensor_{Temperature} = Base_{Temperature} \times Sine\_Value \times Noise \quad (1)$$

The generation formula Eq. (1) was based the base temperature (*Base_Temperature*) was calculated based on the sensor ID value and formed the starting point for each sensor. For example, for a sensor with ID 0, the base temperature started at 20.0 ℃, and for a sensor with ID 1, it started at 21.0 ℃. This decision was made in order to demonstrate the diversity of temperature values for each sensor individually for visual display in combined graphs.

Eq. (2) was calculated based on a sinusoidal function that takes into account the frequency of changes depending on time.

$$\text{Sine\_}Value = \text{A} \times sin(\frac{2\pi \times t}{T}) \quad (2)$$

A—the amplitude of temperature fluctuations. In this case, the amplitude was set to 5.0 ℃, which means that the temperature will fluctuate 5 degrees above and below the base value. *t*—the current time, expressed in hours. It is calculated by dividing the timestamp of the current time by 3600 seconds (the number of seconds in one hour). *T*—the oscillation period, which is 24 h. This means that the temperature will be repeated every 24 h, simulating daily changes. $sin(\frac{2\pi \times t}{T})$—a sinusoidal function that creates smooth periodic oscillations. In this formula, it sets the daily cycle of temperature fluctuations. At the time corresponding to morning and evening, the value of the sine wave will be near zero, and in the middle of the day and night it will reach extreme values. To generate more realistic synthetic data, the noise was modeled using a Gaussian (normal) distribution with a mean of 0 and a standard deviation of 0.5 ℃.

This study assumes that the performance observed in the controlled environment reflects real-world scenarios. However, certain factors, such as hardware variability and network latency, were not considered. Additionally, the synthetic datasets, while representative of typical IIoT workloads, may not capture all complexities of specific industrial applications. The complete benchmarking framework, including data generation scripts and database configurations, is available upon request to facilitate reproducibility. Previously published methods, such as database-specific optimizations, are referenced in the respective sections of this paper.

D. Existing Constraints and Future Extension Directions

This study was intentionally scoped to evaluate the core performance metrics of write throughput and analytical query latency under a high-ingestion workload. It is assumed that the performance observed in this controlled environment is indicative of real-world behavior, though factors like network latency were not considered. The benchmark focused on complex analytical queries, as these are often bottlenecks in IIoT systems. A comparison of simple point-query performance was not included and remains an area for future investigation. Furthermore, this study did not evaluate other important non-functional requirements. A comprehensive analysis of resource utilization (CPU, memory, and storage costs for indexing), elasticity, high availability, and fault tolerance would be valuable extensions to this work. Future research should also explore the practical implementation of the proposed hybrid architecture. This includes evaluating middleware solutions (e.g., Apache Kafka, NiFi) for orchestrating data routing between an ingestion-optimized database like InfluxDB and an analytics-optimized database like TimescaleDB.

III. Results

Benchmarks for data write and read operations were conducted to evaluate the performance of InfluxDB and TimescaleDB databases. The following benchmark extends existing evaluations of InfluxDB and TimescaleDB by demonstrating its stability under sustained high-throughput conditions with real-world IIoT workloads. The results of each database are shown in Figs. 5 and 6. The results obtained make it possible to identify differences in data processing speed between these systems, which is shown in Figs. 7 and 8.

A. Write Performance

The graph in Fig. 5 highlights the robust performance of InfluxDB under a continuous high-frequency data stream, achieving an average write speed of approximately 310,000 records per second. This is evidenced by the stable

plateau observed after an initial increase in write speed, with values consistently ranging between 300,000 and 320,000 records per second. The lack of significant fluctuations demonstrates InfluxDB's reliability in processing large volumes of real-time data, making it an ideal solution for time-series use cases in IIoT environments.

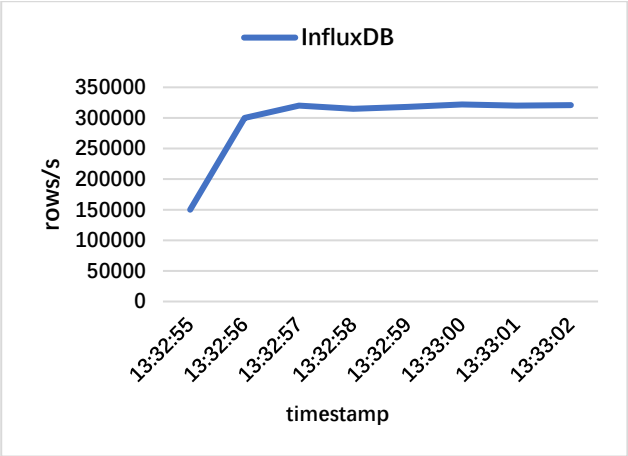

Fig. 5. Write speed over time in InfluxDB.

Compared to existing literature on time-series databases, these results underscore the efficiency of InfluxDB's architecture, particularly its Write-Ahead Log (WAL) mechanism and time-sharded storage. This aligns with findings by other studies, which emphasize the database's optimization for high-speed write operations. Such performance makes InfluxDB a suitable choice for scenarios requiring real-time data ingestion, such as IIoT systems and predictive maintenance in digital twins. The ability of InfluxDB to sustain a stable write speed of 310,000 records per second under high load conditions directly supports its applicability in time-critical IIoT systems and digital twins. This performance ensures seamless real-time data collection, which is crucial for predictive maintenance, process optimization, and decision-making. By reliably handling such high data volumes, InfluxDB minimizes potential bottlenecks in high-frequency data environments, ensuring operational efficiency in industries like manufacturing and energy.

This result demonstrates that InfluxDB excels in scenarios prioritizing write throughput. Nevertheless, when analytical complexity or large-scale querying is required, systems like TimescaleDB might offer complementary advantages, as demonstrated in later sections. Combining the strengths of different database systems could further enhance the flexibility and capability of digital twin architectures.

Fig. 6 illustrates a gradual decline in TimescaleDB's write speed from an initial rate of 6000 records per second to a low of 4500 records per second. This performance decrease, while steady, suggests resource-intensive processes such as chunk creation and index maintenance that are inherent to TimescaleDB's architecture. The system stabilizes slightly toward the end, indicating its ability to maintain functionality under stress while managing internal optimizations.

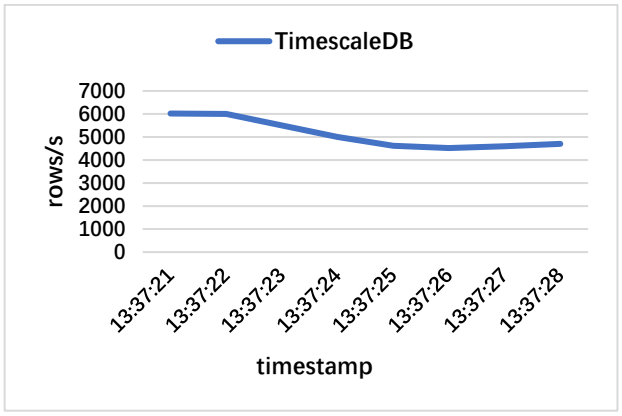

Fig. 6. Write speed over time in TimescaleDB.

The slight recovery in write speed at the end indicates that TimescaleDB can maintain operational stability despite temporary performance drops. This suggests potential optimization mechanisms in play, such as periodic garbage collection or adaptive resource management. The observed decline in TimescaleDB's write performance underscores its limitations in handling large-scale, continuous data ingestion. However, its stability at lower write speeds makes it a reliable choice for applications where analytical capabilities are prioritized over raw ingestion speed. This insight highlights the need for careful consideration of database choice based on the specific requirements of a digital twin system, suggesting TimescaleDB's role in scenarios focused on long-term data analysis rather than real-time monitoring.

These results underline that while TimescaleDB provides consistent operation, it may not be ideal for scenarios requiring sustained high-frequency data writes. Instead, its strengths are likely better leveraged in applications where the focus is on querying and analyzing time-series data rather than purely high-throughput data ingestion. This makes TimescaleDB a suitable choice for use cases such as monitoring systems, financial analytics, and predictive maintenance, where efficient querying and aggregation are more critical than extreme write performance.

Fig. 7 reveals a stark contrast in write performance between InfluxDB and TimescaleDB when handling a dataset of 50,000 rows. InfluxDB achieves a write speed of 150,000 rows per second, significantly outperforming TimescaleDB's 3000 rows per second. This difference underscores InfluxDB's architectural optimization for high-throughput ingestion, whereas TimescaleDB's SQL-based design adds overhead that limits its write efficiency. InfluxDB's superior performance can be attributed to its time-sharded architecture and the use of a Write-Ahead Log (WAL), which are optimized for high-throughput time-series data ingestion. These features allow InfluxDB to handle large volumes of sequential data efficiently, making it a suitable choice for applications requiring rapid data ingestion.

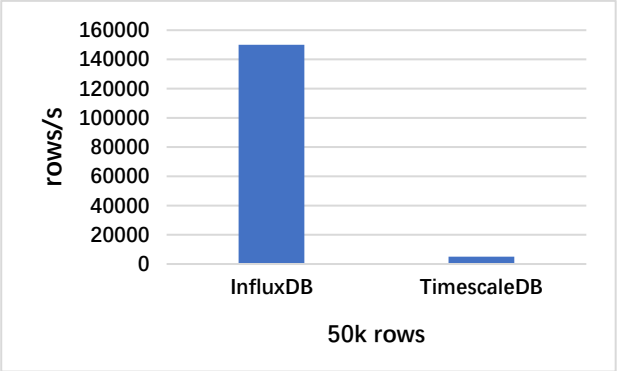

Fig. 7. The results of testing the program in the speed of writing to the database with a volume of 50,000 rows.

In contrast, TimescaleDB, as an extension of PostgreSQL, processes timestamps within the context of relational database structures. This integration adds overhead, such as indexing and chunk management, which impacts its ability to sustain high write speeds. While this design supports complex querying and long-term data analysis, it limits TimescaleDB's performance in write-intensive scenarios. The stark performance difference between InfluxDB and TimescaleDB illustrates the trade-offs in database design for IIoT and digital twin systems. While InfluxDB excels in write-intensive scenarios, TimescaleDB's slower performance points to its reliance on additional functionalities such as SQL processing. These findings emphasize the importance of aligning database selection with system priorities—whether it's high-throughput ingestion or advanced analytics.

These results underline the strengths of each database for different use cases. InfluxDB is well-suited for environments requiring high-frequency data recording, such as real-time monitoring systems. On the other hand, TimescaleDB, with its SQL capabilities, may be more advantageous for applications emphasizing analytics and historical data querying over raw write performance.

B. Query Performance

Fig. 8 compares the query performance of InfluxDB and TimescaleDB for a dataset of 250,000 rows. TimescaleDB processes queries in 40 ms on average, outperforming InfluxDB's 150 ms. This demonstrates the efficiency of TimescaleDB's hypertable design and SQL optimization, which focus query execution on relevant data chunks and enable faster retrieval times, especially for analytical queries.

InfluxDB, on the other hand, processes queries in 150 ms. While this performance is acceptable, it is notably slower than TimescaleDB for the same data volume. The decrease in performance with larger datasets may be linked to InfluxDB's time-sharded architecture, which, while optimized for write operations, can introduce overhead during read operations when accessing specific subsets of data. TimescaleDB's superior query performance at 40 ms per query demonstrates its value in applications requiring rapid data retrieval and analysis, such as operational reporting or historical data trend analysis in digital twins. Conversely, InfluxDB's slower query speed suggests it is better suited for real-time monitoring, where write speed takes precedence. This distinction highlights the complementary roles these databases can play in hybrid architectures.

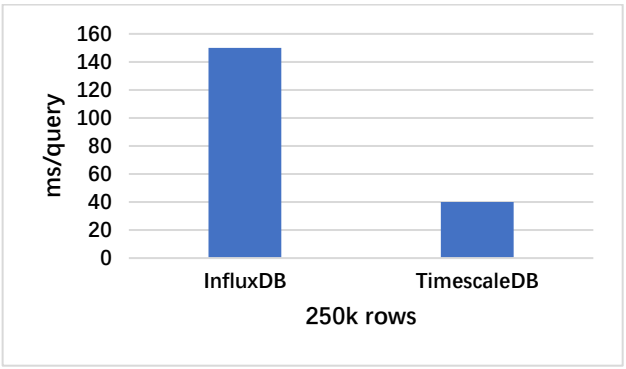

Fig. 8. The results of testing the program in the time spent on processing 1 data select query.

These results highlight the contrasting strengths of the two databases. TimescaleDB excels in read-intensive scenarios requiring efficient data sampling and complex analytical queries, making it well-suited for applications such as historical data analysis and reporting in digital twin systems. Conversely, InfluxDB, with its focus on high write throughput, is better suited for real-time monitoring where frequent read operations are less critical.

IV. Discussion

The results of this study support the initial hypothesis that no single database excels at both high-speed data ingestion and complex analytics for IIoT and digital twin systems. Instead, InfluxDB and TimescaleDB demonstrate a clear trade-off between write-intensive and read-intensive performance, with each being suited to different aspects of a digital twin architecture.

While MongoDB plays a crucial role in Digital Twin architecture for managing entity data, as seen in the OpenTwins framework, it was intentionally excluded from the performance benchmark. Its document-based architecture is optimized for flexible, semi-structured data representing digital twin entities and their complex relationships, not for high-throughput time-series ingestion. Therefore, a direct performance comparison against specialized time-series databases for this specific task would not be a meaningful evaluation of their distinct and complementary functions within a hybrid system.

Similarly, other well-regarded time-series databases such as Open Time Series Database (OpenTSDB) were not included in this benchmark. The primary goal of this study was not to provide an exhaustive survey of all available technologies, but rather to perform a deep comparative analysis of two leading databases with distinct architectural philosophies—InfluxDB and TimescaleDB—that are commonly considered for IIoT applications. This focused comparison allowed us to directly evaluate the trade-offs between a purpose-built time-series solution and a PostgreSQL-based extension, which is central to our recommendation of a hybrid architecture. Future work could certainly extend this

benchmark to include a wider array of databases to provide a broader market overview.

InfluxDB's superior write performance, peaking at over 300,000 records per second, is consistent with findings from other studies that highlight its architectural strengths, such as the Write-Ahead Log (WAL) and time-sharded storage. This makes InfluxDB an ideal choice for the data ingestion layer of a digital twin, where it can reliably handle high-frequency data streams from thousands of sensors in real-time. This capability is critical for applications like real-time monitoring and immediate operational feedback.

Conversely, TimescaleDB's slower write speed can be attributed to the overhead associated with its relational foundation, including indexing and chunk management within its hypertable structure. While this limits its suitability for extreme data ingestion scenarios, these very features are what enable its superior query performance. An average query time of 40 ms on a large dataset makes TimescaleDB highly effective for the analytical layer of a digital twin. It is well-suited for tasks that require complex querying, historical data analysis, and generating insights for predictive maintenance or process optimization.

The performance differences underscore the importance of aligning database selection with specific system priorities. For a digital twin system requiring both real-time data capture and deep analytics, a hybrid architecture is the most effective solution. In such a setup, InfluxDB could serve as the primary time-series data store for incoming sensor data, while TimescaleDB could be used for aggregated data, long-term storage, and as the backend for analytical dashboards and reporting tools. This approach leverages the strengths of both technologies, creating a more robust and capable system than a single-database solution.

V. Conclusion

This study provides a quantitative comparison of InfluxDB and TimescaleDB, revealing their distinct strengths for the data persistence layer of Digital Twin and IIoT applications. The benchmark results offer a significant contribution by providing specific performance metrics: InfluxDB excels at high-throughput data ingestion, with demonstrated write speeds of over 300,000 records per second, making it ideal for the real-time data synchronization required by a Digital Twin. In contrast, TimescaleDB is superior for analytics, executing complex queries in an average of 40 ms, making it suitable for the analytical and simulation components of a DT architecture. These findings provide clear, data-driven evidence that a hybrid database architecture is the most effective approach for systems that demand both high-speed data capture and robust analytics.

This research moves beyond theoretical recommendations by presenting concrete data that empowers engineers to design more efficient and reliable Digital Twin systems. By adopting a hybrid model—using InfluxDB for real-time ingestion and TimescaleDB for analytics—industries can build scalable and future-proof data infrastructures that form the foundation of a high-fidelity Digital Twin.

Future work could extend these benchmarks to include other emerging database technologies or evaluate the performance of middleware designed to manage data flow in such hybrid systems. Ultimately, this study's primary contribution is the technological clarification and new data that validate a hybrid database strategy, advancing the practical implementation of high-performance, data-driven Digital Twins.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

BA conceptualized the study, supervised the project, and wrote the initial draft of the paper. AA made the main contribution to this research; he developed the benchmarking software, conducted the experiments, collected and analyzed the performance data. MK provided methodological guidance and reviewed the manuscript. GA provided overall supervision and project administration. TI assisted with the experimental setup and data collection. DZ contributed to the research conceptualization and manuscript review. All authors have read and approved the final version of the manuscript.

Funding

The article was supported by the project from the Ministry of Science and Higher Education of the Republic of Kazakhstan, BR24992975: "Development of a Digital Twin of a Food Processing Enterprise Using Artificial Intelligence and IIoT Technologies".

References

[1] D. Galar and U. Kumar, "Digital twins: Definition, implementation and applications," in *Advances in Risk-Informed Technologies*, Singapore: Springer Nature Singapore, 2024, ch. 7, pp. 79–106.

[2] Y. Lu, C. Liu, I. Kevin *et al.*, "Digital twin-driven smart manufacturing: Connotation, reference model, applications and research issues," *Robotics and Computer-Integrated Manufacturing*, vol. 61, 101837, 2020.

[3] Y. Jiang, S. Yin, K. Li *et al.*, "Industrial applications of digital twins," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 379, no. 2207, 20200360, 2021.

[4] T. Yu, Z. Li, O. Hashash *et al.*, "Internet of federated digital twins: Connecting twins beyond borders for society 5.0," *IEEE Internet of Things Magazine*, vol. 7, no. 5, pp. 64–71, 2024.

[5] M. P. D. Mudiyanselage and H. Sellahewa, "Digital twins as a framework for IoT applications: A review," in *Proc. 2023 7th SLAAI International Conf. on Artificial Intelligence(SLAAI-ICAI)*, 2023, pp. 1–6.

[6] C. Stergiou, K. E. Psannis, B. G. Kim *et al.*, "Secure integration of IoT and cloud computing," *Future Generation Computer Systems*, vol. 78, pp. 964–975, 2018.

[7] J. F. Yao, Y. Yang, X. C. Wang *et al.*, "Systematic review of digital twin technology and applications," *Visual Computing for Industry, Biomedicine, and Art*, vol. 6, 10, 2023.

[8] National Academies of Engineering, National Academies of Sciences, Engineering, and Medicine, Division on Engineering and Physical Sciences. (2024). Foundational research gaps and future directions for digital twins. Washington. [Online]. Available:

https://nap.nationalacademies.org/catalog/26894/foundational-research-gaps-and-future-directions-for-digital-twins
[9] H. Xu, J. Wu, Q. Pan *et al.*, "A survey on digital twin for industrial internet of things: Applications, technologies and tools," *IEEE Communications Surveys and Tutorials*, vol. 25, no. 4, pp. 2569–2598, 2023.
[10] Y. Zhang, L. Fang, H. Deng *et al.*, "Recent advances and future perspectives of digital twins," in *Proc. 2023 IEEE International Conf. on Control, Electronics and Computer Technology (ICCECT)*, 2023, pp. 1563–1566.
[11] A. A. Mirani, G. Velasco-Hernandez, A. Awasthi *et al.*, "Key challenges and emerging technologies in industrial IoT architectures: A review," *Sensors*, vol. 22, no. 15, 5836, 2022.
[12] H. Wu, P. Ji, H. Ma, *et al.*, "A comprehensive review of digital twin from the perspective of total process: Data, models, networks and applications," *Sensors*, vol. 23, no. 19, 8306, 2023.
[13] A. Fuller, Z. Fan, C. Day *et al.*, "Digital twin: Enabling technologies, challenges and open research," *IEEE Access*, vol. 8, pp. 108952–108971, 2020.
[14] I. A. Arin, Meyliana, H. L. H. S. Warnars *et al.*, "A systematic literature review of recent trends and challenges in digital twin implementation," in *Proc. 2023 10th International Conf. on ICT for Smart Society (ICISS)*, 2023, pp. 1–10.
[15] B. Shah, P. M. Jat, and K. Sasidhar, "Performance study of time series databases," arXiv preprint, arXiv:2208.13982, 2022.
[16] F. Barez, P. Bilokon, and R. Xiong, "Benchmarking specialized databases for high-frequency data," arXiv preprint, arXiv:2301.12561, 2023.
[17] A. Lamb. (October 2022). On InfluxData's new storage engine. Q&A with Andrew Lamb. *ODBMS*. [Online]. Available: https://www.odbms.org/2022/10/on-influxdata-new-storage-engine-qa-with-andrew-lamb/
[18] N. Tran, P. Dix, A. Lamb *et al.* (June 2023). InfluxDB 3.0: System architecture. *influxdata*. [Online]. Available: https://www.influxdata.com/blog/influxdb-3-0-system-architecture/
[19] TimescaleDB: Rearchitecting a SQL database for time-series data. *DataCouncil*. [Online]. Available: https://www.datacouncil.ai/talks25/timescaledb-rearchitecting-a-sql-database-for-time-series-data
[20] C. Diwadkar. (June 2024). Comprehensive comparison between TDengine and TimescaleDB. *TDengine*. [Online]. Available: https://www.tdengine.com/blog/comprehensive-comparison-between-tdengine-and-timescaledb/
[21] TimescaleDB vs Amazon timestream: 6,000x Higher Inserts, 5-175x faster queries, 150-220x cheaper. *Timescale Blog*. [Online]. Available: https://www.timescale.com/blog/timescaledb-vs-amazon-timestream-6000x-higher-inserts-175x-faster-queries-220x-cheaper
[22] PostgreSQL + TimescaleDB: 1,000x faster queries, 90% data compression, and much more. *TimescaleDB*. [Online]. Available: https://www.timescale.com/blog/postgresql-timescaledb-1000x-faster-queries-90-data-compression-and-much-more
[23] F. Reichenbach and M. Yellapantula. (January 2023). Digital twin data middleware with AWS and MongoDB. *AWS*. [Online]. Available: https://aws.amazon.com/blogs/industries/digital-twin-data-middleware-with-aws-and-mongodb/
[24] Scalable IoT projects with MongoDB: Gaining value from IoT & digital twins. *MongoDB*. [Online]. Available: https://www.mongodb.com/resources/solutions/use-cases/webinar-scalable-iot-projects-with-mongodb-gaining-value-from-iot-digital-twins
[25] Y. Fan, J. Yang, J. Chen *et al*., "A digital-twin visualized architecture for flexible manufacturing system," *Journal of Manufacturing Systems*, vol. 60, pp. 176–201, 2021.
[26] A. Khdoudi, T. Masrour, I. E. Hassani *et al.*, "A deep-reinforcement-learning-based digital twin for manufacturing process optimization," *Systems*, vol. 12, no. 2, 38, 2024.
[27] E. Negri, L. Fumagalli, and M. Macchi, "A review of the roles of digital twin in CPS-based production systems," *Procedia Manufacturing*, vol. 11, pp. 939–948, 2017.
[28] J. Robles, C. Martín, and M. Díaz, "OpenTwins: An open-source framework for the development of next-gen compositional digital twins," *Computers in Industry*, vol. 152, 104007, 2023.
[29] T. H. J. Uhlemann, C. Lehmann, and R. Steinhilper, "The digital twin: Realizing the cyber-physical production system for Industry 4.0," *Procedia CIRP*, vol. 61, pp. 335–340, 2017.
[30] Q. Qi and F. Tao, "Digital twin and big data towards smart manufacturing and industry 4.0: 360 degree comparison," *IEEE Access*, vol. 6, pp. 3585–3593, 2018.
[31] A. E. Saddik, "Digital twins: The convergence of multimedia technologies," *IEEE Multimedia*, vol. 25, no. 2, pp. 87–92, 2018.
[32] db-benchmarks. *GitHub*. [Online]. Available: https://github.com/rtzgod/db-benchmarks
[33] N. Togashi and V. Klyuev, "Concurrency in Go and Java: Performance analysis," in *Proc. 2014 4th IEEE International Conf. on Information Science and Technology*, 2014, pp. 213–216.
[34] A. A. Donovan and B. W. Kernighan, *The Go Programming Language*, Boston: Addison-Wesley Professional, 2015.
[35] J. Mostafa, S. Wehbi, S. Chilingaryan *et al.*, "SciTS: A benchmark for time-series databases in scientific experiments and industrial internet of things," in *Proc. 34th International Conf. on Scientific and Statistical Database Management*, 2022, 12.