

TECHNIQUES IN REAL-TIME BIG DATA ANALYTICS: AN APPRAISAL OF ADVANTAGES, LIMITATIONS, AND APPLICATIONS

*Saliyu Angulu Rakiya and M.O. Adenomon

Centre for Cyberspace Studies, Nasarawa State University, Keffi, Nigeria

*Corresponding Author Email Address: rakilira@gmail.com

ABSTRACT

The rapid growth of connected devices, social media, and digital services has produced unprecedented data volumes, driving a shift from batch processing to real-time analytics (RTA). RTA emphasizes velocity, enabling data to be ingested, processed, and analyzed as it arrives to deliver immediate insights and business value. This article reviews major RTA techniques, including stream processing (SP), complex event processing (CEP), query- and rule-based analytics, programmatic platforms, anytime algorithms, and hybrid edge-cloud architectures. The paper assesses stream processing frameworks such as Apache Flink, Kafka, and Storm for scalability, fault tolerance, and latency, with Flink consistently achieving sub-100-millisecond performance for complex tasks. CEP is highlighted for detecting event patterns and correlations, supporting applications in fraud detection, predictive maintenance, and smart environments. Anytime algorithms are examined for their ability to provide progressively refined results, enabling rapid decision-making in time-sensitive contexts. The integration of machine learning into streaming pipelines, edge deployment strategies guided by the latency-bandwidth-compute triangle, and emerging hybrid architectures are also discussed. Three developmental trends are identified: evolution from batch to stream-native systems, convergence of batch and stream paradigms, and fusion of machine learning with RTA. Findings underscore the need to balance latency, cost, and complexity when selecting techniques for specific use cases.

Keywords: Real-time data analytics, Complex Event Processing, Apache Flink, Apache Kafka, anytime algorithms, machine learning integration

INTRODUCTION

The advent of ubiquitous connectivity, social media ecosystems, and digital service platforms has fundamentally transformed the digital landscape, giving rise to an overwhelming proliferation of information (Joseph, 2025) (Mehmood *et al.*, 2025). This rapid expansion of data generation, distinguished by its extraordinary volume, velocity, and variety, has precipitated a paradigm shift in the architecture and governance of computational infrastructures (Chen *et al.*, 2023). Contemporary systems must therefore address the formidable challenge of ingesting and processing continuous data streams in real time, driven by the widespread deployment of IoT devices, the ubiquity of social media platforms, and the proliferation of digital services that collectively generate an immense torrent of information (Sharma, 2026). Real-time analytics has emerged as a distinct branch of big data analytics that emphasizes the velocity aspect of big data, in which data is prepared, processed, and analyzed as it arrives to generate

insights and create business value in near real time (Chen *et al.*, 2023). Traditional batch-processing frameworks, which analyze data at scheduled intervals and produce insights that may be hours or days old, struggle to meet these demands, driving the need for advanced real-time analytics solutions (Premalatha, 2026; Manjusha & Kumar Sharma, 2026). The difference between real-time and batch processing is not merely a matter of speed; it is the difference between detecting a fraud attempt before a transaction clears and discovering it in a morning report, or between scheduling equipment maintenance before a failure and responding to an outage that has already stopped production (Chen *et al.*, 2023). The need for real-time data processing and analysis has led to two important paradigms: generic Stream Processing (SP) and Complex Event Processing (CEP) (Kyrama & Gounaris, 2026). While Stream Processing focuses on analyzing continuous data streams in real-time to extract insights, Complex Event Processing specializes in identifying meaningful event patterns and correlations, such as security breaches, machine malfunctions, or financial fraud (Kyrama & Gounaris, 2026). This article provides a comprehensive examination of these primary techniques, analyzing each technique's advantages, limitations, and applications across industries.

Real-Time Analytics versus Batch Analytics

The distinction between real-time analytics and traditional batch analytics is fundamental to understanding the value proposition of real-time techniques (Haapala, 2026; Patil & Moantri, 2026). Batch analytics processes data on a schedule such as hourly, daily, or weekly, and works well for historical questions such as quarterly trends or weekly revenue (see Figure 1). Real-time analytics processes data continuously or near-continuously and supports active questions such as whether to trigger an alert, block an action, or update a user experience (Chen *et al.*, 2023). Key differences include latency, best use cases, business consequences, infrastructure complexity, and cost trade-offs (Al Khaldy & Hamarsheh, 2026). Batch analytics typically exhibits latency measured in scheduled intervals such as hourly, daily, or weekly, making it suitable for historical analysis, financial reporting, long-term trends, and scheduled summaries (El-Kafrawy & El-Amin, 2026; Manjusha & Kumar Sharma, 2026). The business consequence of batch analytics is that it helps teams understand what happened after the fact (Modalavalasa, 2026; Haapala, 2026). In contrast, real-time analytics delivers insights with seconds, milliseconds, or near-real-time latency, making it best for active decisions, alerts, monitoring, live dashboards, and embedded product experiences (Önden, 2026; Li, 2026). The business consequence is that real-time analytics helps teams respond before the moment passes (Chen *et al.*, 2023).

Infrastructure complexity is generally higher for real-time analytics because ingestion, processing, and delivery must remain continuously available, whereas batch analytics can be processed in scheduled jobs with lower infrastructure requirements (Parmar, T., 2025; Shermey & Saranya, 2025). The cost tradeoff is that real-time analytics can become expensive if it processes high-volume events inefficiently. In contrast, batch analytics is usually easier to control but less useful for time-sensitive decisions (Chen *et al.*, 2023; Siddiqui & Qureshi, 2025).

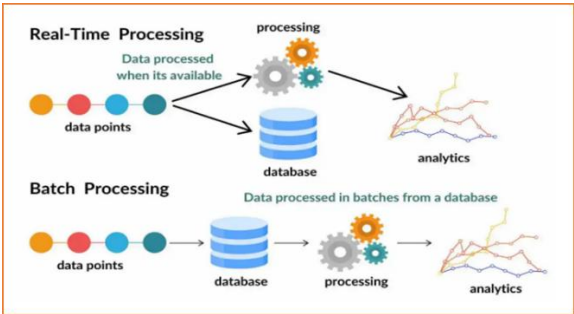


Figure 1: Real-Time and Batch Processing (Chen *et al.*, 2023)

The principal differences between the two paradigms can be summarized as follows:

Table 1a: Differences between Batch Analytics and Real-Time Analytics (Patil & Moantri, 2026) (Khattach *et al.*, 2025)

Dimension	Batch Analytics	Real-Time Analytics
Data processing	Periodic	Continuous or near-continuous
Typical latency	Minutes to days	Milliseconds to seconds
Primary objective	Retrospective understanding	Immediate decision support
Typical applications	Reporting, historical analysis	Fraud detection, monitoring, alerts
Infrastructure	Scheduled processing	Continuously available infrastructure
Data arrival	Accumulated datasets	Continuous streams
Decision timing	After events	During or immediately after events
Cost structure	Generally easier to control	Potentially higher due to continuous operation
Fault tolerance	Job-level recovery	Continuous state and fault recovery
Main challenge	Large-scale computation	Latency, state, availability, and correctness

The choice between batch and real-time analytics should not be viewed as binary. Many modern systems combine both paradigms. Historical data may be processed using batch analytics for model training and long-term analysis, while real-time streams support online inference, monitoring, and immediate decision-making. This convergence creates a unified analytical architecture in which batch and streaming processes complement each other.

Review Objectives

The principal objective of this review is to critically examine major techniques used in real-time data analytics and establish a structured framework for understanding their relative advantages, limitations, and application domains.

The specific objectives are to:

1. Examine the fundamental principles of major real-time analytics techniques.
2. Compare query-based, rule-based, stream-processing, CEP, anytime, ML/AI, and edge-cloud approaches.
3. Analyze representative real-time stream-processing frameworks.
4. Identify the principal performance dimensions used to evaluate real-time systems.
5. Examine the integration of machine learning and artificial intelligence with streaming pipelines.
6. Analyze the role of edge, fog, cloud, and hybrid architectures.
7. Identify persistent research challenges and gaps.
8. Propose future research directions for intelligent and adaptive real-time analytics.

Review Questions

The following questions guide the review:

- RQ1: What are the principal techniques used for real-time data analytics?
RQ2: How do these techniques differ in terms of latency, scalability, computational requirements, and analytical capability?
RQ3: What are the relative advantages and limitations of major stream-processing platforms?
RQ4: How does Complex Event Processing differ from generic stream processing?
RQ5: What role do anytime algorithms play in time-constrained analytics?
RQ6: How can machine learning and AI be integrated into streaming architectures?
RQ7: How do edge-cloud architectures influence latency, bandwidth, privacy, scalability, and energy consumption?
RQ8: What research gaps must be addressed to develop next-generation real-time analytics systems?

Classification of Data Stream Analytics Platforms

Data stream analytics platforms fall into three principal types (see Table 1b) based on how they process and analyze streaming data: query-based, rule-based, and programmatic platforms (Chen *et al.*, 2023). Query-based platforms, including products such as Esper, Flink SQL, SAP HANA Streaming Analytics, StreamSQL, and Azure Streaming Analytics, express low-level aggregation on defined event types using SQL-based query languages. Rule-based platforms are divided into production rule systems like Drools Fusion and Business Events, which use IF-THEN rules to respond to the environment, and Event-Condition-Action rule systems like IBM Streams, which facilitate reactions to events in event-driven architectures (Chen *et al.*, 2023). Programmatic platforms such as Flink, EventSwarm, Apana, and StreamBase provide comprehensive functionality for processing complex events through programming interfaces and APIs.

Table 1b: Classification of Data Stream Analytics Platforms (Igulu *et al.*, 2025) (Abashes *et al.*, 2025)

Category	Products	Description
Query-based	Esper, Flink SQL, SAP HANA Streaming Analytics, StreamSQL, Azure Streaming Analytics	Express low-level aggregation on defined event types; SQL-based query languages
Rule-based: Production Rules	Drools Fusion, BusinessEvents	IF-THEN rules about behavior responding to environment
Rule-based: ECA Rules	IBM Streams	Event-Condition-Action rules for event-driven architecture
Programmatic	Flink, EventSwarm, Apana, StreamBase	Comprehensive functionality for processing complex events

Review Methodology

This appraisal used a systematic approach to ensure transparency and replicability. The methodology follows the principles of narrative synthesis adapted for technical reviews in computing and information systems.

Search Strategy

A systematic search was performed across multiple academic databases, including IEEE Xplore, ACM Digital Library, ScienceDirect, SpringerLink, and Google Scholar.

Inclusion and Exclusion Criteria

The inclusion criteria used in this study include:

- Peer-reviewed journal articles, conference papers, and book chapters
- Papers published between 2015 and 2026 to ensure currency
- Focused on real-time data analytics techniques, frameworks, or applications
- Papers providing empirical evaluation, comparative analysis, or comprehensive review

The exclusion criteria in this study include:

- Papers focused solely on batch processing without real-time considerations
- Non-peer-reviewed articles, blogs, or technical reports
- Papers not available in full text
- Papers focused exclusively on hardware implementation without algorithmic discussion

Analytical Framework

The included papers were analyzed using thematic analysis. Each paper was categorized based on:

- Technique Type: Query-based, rule-based, stream processing, CEP, anytime algorithms, ML/AI integration, edge-cloud architectures

- Performance Dimensions: Latency, throughput, scalability, fault tolerance, resource consumption
- Application Domain: Finance, healthcare, IoT, smart cities, industrial monitoring
- Advantages and Limitations: As reported by the authors

The synthesis involved identifying common themes, tracing the evolution of techniques over time, and identifying research gaps and future directions.

Query-Based Data Stream Analytics

Query-based data stream analytics platforms offer strong expressiveness for low-level aggregation on defined event types, and their SQL basis makes them easy for most application developers to learn (Chen *et al.*, 2023). These platforms typically use Event Processing Languages (EPLs) that let users express complex queries over streaming data using familiar SQL-like syntax (Langhi, 2025) (Chandra, 2025). The SQL foundation makes query-based platforms accessible to application developers with existing database experience and provides strong support for low-level aggregation operations on defined event types.

Advantages

Query-based platforms are easy to learn, as the SQL foundation makes them accessible to application developers with existing database experience (Chen *et al.*, 2023). They provide good expressiveness for low-level aggregation operations on defined event types. Standardizing SQL-like query syntax reduces the learning curve for new users and enables rapid development of streaming analytics applications (Medina-Bulo *et al.*, 2025; Lin *et al.*, 2025).

Limitations

Query-based platforms have drawbacks in event-pattern abstraction, and some languages cannot express dynamic sliding windows (Chen *et al.*, 2023). Additionally, tuple-based event-type models often struggle with nested data found in JSON or XML payloads (Hassan, 2025). These limitations restrict the complexity of patterns they can express and may require workarounds for advanced use cases (Harditya *et al.*, 2026).

Applications

Query-based platforms suit scenarios that require low-level aggregation and analysis of streaming data, such as real-time monitoring dashboards, IoT data aggregation, and operational analytics (Chandra *et al.*, 2025). They are particularly effective when the analytical requirements are well-defined and can be expressed using SQL-like constructs (Baccouri & Abdellatif, 2025).

Rule-Based Data Stream Analytics

Rule-based data stream analytics platforms are better suited to expressing conditional event patterns than query-based platforms (Chen *et al.*, 2023). Two principal types of rule-based techniques have been used: production rules and Event-Condition-Action rules.

Production Rules and Ripple-Down Rules

Production rule data stream analytics platforms consist of a set of IF-THEN rules about behavior that respond to the environment: if the condition is met, the action is asserted (Chen *et al.*, 2023; Babadzhanov *et al.*, 2025). As a common technique in expert systems since the 1980s, production rules have provided an

alternative to artificial intelligence for knowledge representation and business decision-making across a broad range of domains, including pathology and computer system and network troubleshooting. The ripple-down rules (RDR) framework enables incremental learning, with domain experts adding rules when false conclusions occur, without corrupting existing knowledge bases (Chen *et al.*, 2023; Puslecki & Walkowiak, 2025).

Advantages of Production Rules and Ripple-Down Rules

Production rules provide established knowledge representation techniques that the expert systems community understands well. They support business decision-making through automated rule execution and enable incremental learning through the RDR framework (Chen *et al.*, 2023). This makes them particularly valuable in domains where human expertise can be encoded as rules and refined over time.

Limitation

Forward-chaining inference can lead to conflicting rules, requiring careful rule management (Chen *et al.*, 2023). The action has to be modified whenever the condition is updated, creating maintenance overhead. Rule execution results are typically not preserved for future improvements, limiting the ability to learn from past decisions. Production rules find applications in IoT event processing and monitoring, business rule automation, and computer system and network troubleshooting (Das & Rajput 2026).

Event-Condition-Action Rules

Event-Condition-Action rules were proposed to facilitate reactions to events in event-driven architectures and active databases (Das & Rajput 2026). There are three components in an ECA rule: the event, which is the trigger of rule execution and can be a composite event comprised of various event types; the condition, which is the criterion that needs to be satisfied via a logic test before the specified action is taken; and the action, which is the operation to be executed when the condition is met. ECA rules are designed for event-driven architectures and can handle composite events made up of various event types (Chen *et al.*, 2023). They enable automated rule execution through expert system integration and clearly separate event detection, condition evaluation, and action execution. Also, expressing complex event patterns can be challenging with ECA rules (Richards *et al.*, 2025). Managing large rule bases requires sophisticated tooling and careful organization. Full automation requires integration with expert systems, adding architectural complexity. ECA rules detect event pattern occurrences and react to them, execute business logic on incoming event streams, manage active databases, and implement event-driven architectures (Poss & Schöning, 2026).

Stream Processing: Frameworks and Comparative Analysis

Stream Processing has become the standard model for analyzing continuous, rapid information flows from different sources with low latency to produce new data streams and extract knowledge and insights from raw data (Kyrama & Gounaris, 2026). Modern stream processing systems manage real-time data inputs, enabling organizations to detect anomalies, identify emerging patterns, and quickly adapt to changing conditions (Shaik, 2024). Prominent platforms include Apache Flink, Apache Kafka, and Apache Storm, each offering distinct features tailored to specific aspects of stream processing.

Apache Flink

Apache Flink is a distributed stream processing framework designed around continuous data flows. Unlike batch-first systems that bolt on streaming, Flink treats streams as the primary abstraction, with batch processing considered as a bounded stream (StreamKap, 2026). A key advantage is its ability to perform stateful stream processing with precise event-time semantics, which is critical for applications that demand high accuracy and dependability.

Architecture and Operational Characteristics

Flink runs as a cluster with a JobManager (coordinator) and one or more TaskManagers (workers). It stores state in RocksDB on local disk and periodically checkpoints it to a distributed filesystem like S3 or HDFS. Checkpoints are incremental, meaning only changed state gets written, keeping checkpoint overhead low even for multi-terabyte state (StreamKap, 2026). Flink processes events one at a time, not in micro-batches, which gives it single-digit millisecond latency in most configurations. Throughput scales horizontally by adding TaskManagers, with production deployments regularly handling millions of events per second per job.

State Management

Flink offers exactly-once state consistency through aligned and unaligned checkpoints, supports keyed state and operator state, and can manage terabytes of state per job (StreamKap, 2026). Savepoints let users stop a job, modify the code, and resume from the same state, enabling seamless upgrades and maintenance.

SQL Support

Flink SQL is a full SQL layer that compiles down to the same dataflow runtime. Users can create tables, define watermarks for event time, run joins including temporal joins, and write results to sinks all in SQL. The SQL support is mature enough for production use, though complex windowing logic sometimes requires dropping into the Java/Scala DataStream API (StreamKap, 2026). The advantage is that Flink provides a true stream-native architecture with low latency, battle-tested state management at terabyte scale, strong SQL support, an active open-source community, and exactly-once processing guarantees (StreamKap, 2026). Performance benchmarks indicate that Flink consistently achieves sub-100-millisecond latency for complex operations (Shaik, 2024). The steep operational learning curve for self-managed clusters and JVM tuning required for large state workloads present challenges (StreamKap, 2026). Job upgrades with state compatibility need careful planning, and managed offerings can become expensive at scale.

Apache Kafka

Apache Kafka, developed by LinkedIn before becoming open-source, functions as a distributed event streaming platform optimized for high-throughput data ingestion and real-time distribution (Shaik, 2024). Its architecture relies on a replicated log structure to guarantee durability and scalability across multiple nodes. With the ability to handle millions of events per second at sub-millisecond latencies, Kafka has become an essential building block for real-time data pipelines and applications (Javed & Rajumar, 2026).

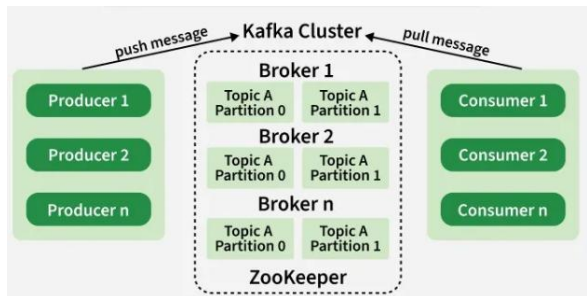


Figure 2: Kafka Architecture (Goswami & Shivaji, 2025)

Advantages

Kafka provides high throughput and can process millions of events per second with sub-millisecond latencies (Goswami & Shivaji). The replicated log structure ensures durability and fault tolerance, and the partitioned architecture allows horizontal scaling across distributed clusters. Kafka Streams facilitates in-stream computations while Kafka Connect enables integration with external systems. Kafka Streams emerges as a compelling solution for lightweight stream processing needs, particularly in Kafka-centric architectures (Shaik, 2024).

Limitations

Configuring and managing Kafka clusters requires specialized expertise. While excellent for ingestion, complex transformations may require additional processing layers (Bhattacharyya *et al.*, 2025). The persistent log structure requires significant storage resources. As Shaik (2024) notes, Kafka Streams provides the most straightforward operational model among major stream processing platforms.

Applications

Kafka is used in high-frequency trading environments where market activity histories must be reconstructed for thorough analysis, IoT data ingestion pipelines in smart city applications collecting traffic patterns, weather conditions, and energy usage data, and social media stream ingestion for sentiment analysis and trend detection (Bolgov, 2025).

Apache Spark Structured Streaming

Apache Spark is a newer-generation framework than Hadoop for large-scale data analytics, with an execution engine that distributes programs across clusters and modules for writing programs in various languages, including Python, SQL, Scala, Java, and R (Yip & Wojtyczka, 2026). Spark Structured Streaming extends the Spark SQL engine to handle streaming data using a micro-batch execution model by default, processing small batches of data at regular intervals (Alam, 2025).

Advantages

Spark offers high scalability with linear scalability characteristics, a comprehensive set of tools including SQL, streaming, ML, and graph processing, and uses random-access memory instead of disks for intermediate data, resulting in performance at least three times faster than Hadoop (Shaik, 2024). The unified batch/streaming API means the same SQL works on both bounded and unbounded data, and the large ecosystem includes MLlib, GraphX, and Delta Lake integration (Streamkap, 2026). Spark Streaming offers unparalleled throughput for large-scale data

processing (Shaik, 2024).

Limitations

The default micro-batch mode provides latency in the hundreds of milliseconds to seconds range, which is sufficient for dashboards and analytics but not for sub-second alerting (Immadisetty, 2025). Spark 3.x introduced a continuous processing mode targeting approximately 1ms latency, but it remains experimental with limited operator support. Checkpoint-based recovery can be slow, and Spark has a heavier resource footprint than stream-native tools. However, Spark is best for organizations already running Spark for batch ETL or analytics that want to add streaming without introducing a new framework, and when combining streaming with ML model scoring or complex analytical queries (Gudduri, 2026).

Table 2: Comparison of Stream Processing Tools (Gudduri, 2026)

Tool	Latency	State Management	SQL Support	Deployment Model	Best For
Apache Flink	Low ms	RocksDB, incremental checkpoints	Flink SQL (full)	Self-managed or managed	Stateful event processing at scale
Kafka Streams	Low ms	RocksDB, changelog topics	None (Java DSL only)	Embedded in JVM apps	Kafka-native microservices
Spark Structured Streaming	100ms+ micro-batch, ~10ms continuous (experimental)	In-memory + checkpoint to HDFS/S3	Spark SQL (full)	Self-managed, Databricks, EMR	Teams already running Spark batch jobs
Apache Storm	Low ms	External only	Limited	Self-managed	Simple, scalable real-time computation

Practical Considerations for Stream Processing Tool Selection

Empirical studies guide tool selection. Shaik (2024) conducted comprehensive benchmarking and found that Flink consistently achieves sub-100-millisecond latency for complex operations, while Spark Streaming offers unparalleled throughput for large-scale data processing, and Kafka Streams provides the most straightforward operational model. The study also found significant feature convergence across these platforms, with each adopting strengths from the others while maintaining its distinct architectural advantages (Shahnawaz & Kumar, 2025).

A comparative study on real-time fraud detection using Kafka with Apache Flink and Apache Spark found that both pipelines were implemented under identical conditions and integrated with a pretrained Random Forest model. Experimental results showed that Apache Spark achieved consistently lower average latency of 0.8 seconds at 10 TPS and 0.9 seconds at 40 TPS compared to Apache Flink, which recorded 1.7 and 1.6 seconds, respectively. Throughput remained stable under load for both frameworks, demonstrating scalability (Shahnawaz & Kumar, 2025). These findings provide empirical insights to support informed decision-

making in selecting stream processing technologies for real-time, mission-critical use cases.

Complex Event Processing

Complex Event Processing is a specialized category of stream processing that focuses on identifying meaningful sequential event patterns in data streams, such as security breaches, machine malfunctions, or financial fraud (Kyrama & Gounaris, 2025). Complex Event Processing systems monitor data streams in real time, filter out irrelevant events and combine incoming events from single or multiple data streams to identify high-level events, i.e., complex events (Kyrama & Gounaris, 2025).

Distinctions Between CEP and SP

Complex Event Processing differs from generic Stream Processing systems in four aspects: data model, language model, processing model, and time model (Kyrama & Gounaris, 2025). Stream Processing systems consider the internal data model as a tuple, i.e., a list of attributes, and all incoming tuples share a common schema, forming a stream of tuples. In Complex Event Processing systems, data arrive as events, i.e., tuples with a timestamp specifying the event generation time. The source generates and attaches the timestamp, which serves as an ordering marker when the system processes the event (Kyrama & Gounaris, 2025).

Table 3: Distinctions Between Complex Event Processing and Generic Stream Processing (Kyrama & Gounaris, 2025)

Aspect	Stream Processing	Complex Event Processing
Data Model	Tuples with common schema	Events with timestamps
Language Model	Low-level APIs or SQL-like	Declarative pattern language
Processing Model	Generic transformations	Pattern detection and sequencing
Time Model	Processing time focus	Event time focus

Advantages and Limitations of CEP

Complex Event Processing systems are designed to capture and process data semantics, using expressive languages to define complex event patterns for detecting specific situations or event combinations (Kyrama & Gounaris, 2025). The timestamp serves as an ordering marker during processing, enabling accurate temporal analysis. Complex Event Processing systems provide sophisticated state management for tracking complex patterns across multiple events and can trigger immediate actions based on detected patterns.

However, pattern detection over long event sequences requires significant memory and computational resources. Compared to general Stream Processing systems, Complex Event Processing systems may struggle to scale to extremely high event volumes. Several advanced Complex Event Processing solutions cannot be transferred to edge environments because they assume abundant, homogeneous resources; assume out-of-order and missing data are managed elsewhere; and assume privacy constraints on data sharing (Kyrama & Gounaris, 2025). Additionally, Kumar and Agarwal (2024) provide a comprehensive survey of rule-based CEP for IoT applications, highlighting challenges in rule adaptability for streaming IoT data and the need for scalable event-driven architectures.

Applications of CEP

Consider a smart home scenario with sensors measuring temperature, humidity, and air quality installed in each room. The data from these sensors are sent to a Complex Event Processing system running on a Raspberry Pi, which is also installed locally. The system is responsible for detecting important patterns in real-time, such as when the temperature is higher than a predefined value, there is a continuous rise of temperature within the last 30 minutes, or the indoor air is too dry and the temperature is too high, indicating that the humidity control system is not working properly (Kyrama & Gounaris, 2025). This Complex Event Processing system can serve as an HVAC control system that triggers an alert or adjusts settings. Complex Event Processing finds applications in smart home environments, fraud detection where it identifies sequences of events that indicate fraudulent activity such as unusual transaction patterns, industrial monitoring where it detects machine malfunctions through patterns in sensor data, and healthcare monitoring where it identifies critical patient conditions through patterns in vital sign data (Kyrama & Gounaris, 2025).

Anytime Algorithms for Real-Time Analytics

Anytime algorithms have gained significant attention in data analytics because they can provide approximate results at any time that improve over time, making them well-suited for quick decision-making (Challa *et al.*, 2026). Anytime algorithms, which can trade computational time for result quality, are increasingly critical for applications requiring rapid, adaptive insights. They are widely used in stock market analysis, fraud detection, sentiment analysis, and weather forecasting (Challa *et al.*, 2026). Traditional data analytics techniques often struggle to process large-scale data, especially in environments with constrained computational resources and processing time. They are often inadequate for handling variable data inter-arrival rates because of their limited speed and lack of flexibility. If used beyond their maximum capability, they would have to process sampled data or buffer unlimited data and eventually get stuck (Asadova *et al.*, 2023).

Characteristics of Anytime Algorithms

Anytime algorithms exhibit four key characteristics: interruptibility, meaning they can be interrupted at any time before completion and provide intermediate, valid approximate results; resource adaptiveness, where the quality of results improves with an increase in allocated resources; incremental learning, enabling progressive refinement as more data becomes available; and scalability, maintaining anytime properties with an increase in data size or complexity (Asadova *et al.*, 2023) (Shamsinejad *et al.*, 2025).

Table 4: Comparison of Non-Anytime Versus Anytime Algorithms (Challa *et al.*, 2026)

Aspect	Non-Anytime Algorithms	Anytime Algorithms
Execution model	Run to completion before producing any output	Can be interrupted at any point to provide best-so-far result
Result availability	Only available after full processing	Progressive refinement; output improves with time
Adaptability to time constraints	Fixed runtime	Adapts to varying time allowances
Handling resource constraints	Fixed computational budget	Adjusts processing based on available resources
Scalability	Requires more hardware	Provides intermediate results, refines over time
Use in real-time decision-making	Cannot return intermediate insights before completion	Well-suited for real-time and streaming environments
Quality of intermediate results	No intermediate result; must wait for final result	Quality improves with increase in time; early results may be coarse but usable

Contract versus Interruptible Anytime Algorithms

Anytime algorithms fall into two categories: contract and interruptible (Challa *et al.*, 2026). A contract algorithm receives resources such as time and computational hardware allocated beforehand and executes within those allotted resources to produce the best possible result. If interrupted before the contract ends, contract algorithms do not guarantee a valid result. An interruptible algorithm, by contrast, is designed to return a result at any point during execution, without preallocated resources. If interrupted, it returns the best solution found so far; if allowed to continue, it refines the result further. For instance, in a fraud detection system, if the algorithm is interrupted before it finishes, it can produce quick fraud alerts; with more processing time, it can refine the accuracy of these alerts as it analyzes more data (Challa *et al.*, 2026). This feature makes interruptible algorithms highly flexible and suitable for environments where computational resources or time availability are uncertain.

In data analytics, anytime algorithms apply to two broader scenarios: mining large databases and mining data streams (Asadova *et al.*, 2023). Anytime algorithms for large datasets produce multiple results with varying levels of approximation, whose accuracy increases as processing time increases, giving users flexibility in trading off computational resources and result accuracy. Anytime algorithms for data analytics include induction-tree-based, kernel-based, ensemble-based, neural-network-based, nearest-neighbors-based, probability-estimation-based, density-based, and frequent itemset mining algorithms.

Performance Metrics for Anytime Algorithms

To evaluate the performance of anytime algorithms, particularly for large datasets, metrics must capture their ability to provide progressively improved solutions as more computational resources become available (Challa *et al.*, 2026). Unlike traditional algorithms, which return a single final result after execution, anytime algorithms can be evaluated by the quality of intermediate results and the speed at which they improve over time. Key metrics include (Chen, 2025):

- Initial Solution Quality under Interruption:** Measures the quality of the first available intermediate solution when the algorithm is interrupted early (Neoaz, 2025).
- Time-Computational Efficiency Trade-off:** Evaluates how efficiently an anytime algorithm transforms available computational resources into quality improvements.
- Scalability:** Measures how well an algorithm maintains its anytime properties with an increase in the size or complexity of data or stream speed (Asadova *et al.*, 2023).

Applications of Anytime Algorithms

Anytime algorithms find applications in stock market analysis and trend prediction, fraud detection in credit card transactions, sentiment analysis in social media analytics, speech recognition in natural language processing, object recognition from images and videos, and path planning in robotics (Challa *et al.*, 2026). For example, in stock market analysis, anytime algorithms can process large-scale data incrementally, providing preliminary insights quickly if interrupted and refining predictions as more resources become available, benefiting both short-term stock investors seeking quick results and long-term stock investors willing to wait for more refined, accurate results.

Recent research has investigated the utilization of anytime algorithms in cloud-based fault detection techniques. Asadova *et al.*, (2023) analyzed the basic concepts of anytime algorithms in existing works as well as their benefits of usage, reviewing various literature and combining them for possible fault recognition and time deliberation. The study showed that anytime algorithms, real-time decision-making algorithms that focus on time deliberation, are particularly valuable for fault detection in cloud systems where timely response is critical (Asadova *et al.*, 2023).

Machine Learning Integration in Streaming Pipelines

Integrating machine learning into stream processing systems has significantly enhanced analytical capabilities in real-time data environments (Shaik, 2024). This convergence enables continuous deployment of machine learning models to evaluate and interpret incoming data streams, which is essential for applications such as predictive maintenance and anomaly detection. Tools like TensorFlow Extended and Apache Beam enable seamless integration of machine learning workflows into streaming pipelines, allowing adaptive training and real-time model assessment. Machine learning models can score incoming events immediately, enabling instant decision-making (Shaik, 2024). For instance, in a dynamic pricing strategy for e-commerce platforms, prices adjust in real time based on supply-demand analytics extracted from continuous customer activity streams. Models can update dynamically using incoming streams, accommodating new trends and shifts in consumer behavior almost instantaneously. Machine learning algorithms can detect subtle patterns that rule-based

systems might miss, and hybrid models that combine multiple AI techniques further enhance predictive performance (Chen *et al.*, 2023).

Advantages of ML Integration

The following are some of the advantages of ML algorithms:

- a. Real-Time Predictions: Models can score incoming events immediately, enabling instant decision-making. For instance, in fraud detection, a pre-trained Random Forest model can be integrated with Kafka and either Flink or Spark to score transactions as they arrive.
- b. Adaptive Learning: Models can be updated dynamically using incoming streams, accommodating new trends and consumer behavior shifts almost instantaneously. This lets systems adapt to changing patterns without manual model retraining.
- c. Enhanced Insights: Machine learning algorithms can detect subtle patterns that rule-based systems might miss, and hybrid models that combine multiple AI techniques further enhance predictive performance (Chen *et al.*, 2023).

Limitations and Challenges

The following are some of the limitations of integrating ML in network systems:

- a. Computational Demands: Running machine learning models in real-time requires significant computational resources. Deep learning models, including multi-layer perceptrons, convolutional neural networks, recurrent neural networks, and long short-term memory networks, demand substantial resources (Chen *et al.*, 2023).
- b. Latency: Complex model inference may introduce unacceptable delays, potentially negating the benefits of real-time processing.
- c. Data Quality: Errors that batch pipelines catch during scheduled validation propagate instantly in streaming systems, requiring quality controls embedded in the ingestion layer itself.
- d. Data Scarcity: In embedded AI applications, the data bottleneck is significantly more pronounced than in mainstream AI, creating label scarcity and sample scarcity challenges.

ML/AI Techniques in Real-Time Analytics

The choice of AI techniques depends on user preference. The following are some of the techniques:

- a. Supervised Learning Techniques: Regression, Random Forest, SVM, and Gradient Boosting are applied to classification, prediction, and fraud detection (Chen *et al.*, 2023). In the fraud detection study, a Random Forest model was selected for integration with Kafka-based real-time pipelines because it performed best offline among five evaluated machine learning classifiers (Menon, 2025).
- b. Unsupervised Learning Techniques: K-Means, DBSCAN, and PCA are applied to clustering, anomaly detection, and dimensionality reduction (Chen *et al.*, 2023).
- c. Deep Learning Models: Convolutional Neural Networks for processing grid-like data such as images, Recurrent Neural Networks for sequential data with temporal

dependencies, Long Short-Term Memory for capturing long-term dependencies in sequences, Gated Recurrent Units as simplified LSTM variants, and Autoencoders for unsupervised learning and dimensionality reduction have been extensively employed (Chen *et al.*, 2023).

Edge-Based Deployment and Hybrid Architectures

The following subsection discusses edge-based deployment and hybrid architectures for network systems.

The Edge-Cloud Design Triangle

Deploying stream processing and Complex Event Processing systems in edge settings addresses latency, security, and privacy concerns while highlighting challenges specific to these systems (Kyrama & Gounaris, 2026). Edge nodes operate under limited resources, unstable connectivity, and high transmission costs (see table 5), often under different ownership, performing local analytics to reduce communication overhead and latency while preserving privacy and security by keeping sensitive data close to its source (Martis, 2025). The latency, bandwidth, and compute constraints form the "design triangle" that helps engineers determine where processing occurs, what data gets prioritized, and how resilient the entire system is under real-world conditions (Martis, 2025). Latency influences whether data is processed immediately on the edge or more slowly via the cloud, with time-sensitive data such as health alerts or anomaly detection requiring low latency and edge computing. Bandwidth determines which data is sent to the cloud, requiring intelligent filtering, local caching, and compression for optimization. Compute resources are less available in edge computing than in the cloud, making it crucial to optimize for lightweight telemetry agents that prioritize the most valuable insights (Martis, 2025).

Table 5: Comparison of Infrastructure Types for Real-Time Analytics

Infrastructure Type	Advantages	Limitations
Single-node	Easily deployed	Limited scalability
Cluster	High-performance, high-availability, scalability	Fixed processing nodes, limited compared to cloud
Cloud-based	Scalability, mobility	Reliance on network connectivity, significant latency
Edge/Fog	Low communication latency, security, privacy	Limited scalability, possible data loss

Advantages of Edge-Based and Hybrid Architectures

The following are the advantages of edge-based and hybrid architectures:

- a. Reduced Latency: Hybrid architectures combine the power of cloud computing and the speed of edge telemetry, enabling ultrafast response times for mission-critical applications (Martis, 2025). Data processed close to the source reduces transmission delays.

- b. **Enhanced Security and Privacy:** Processing sensitive information locally reduces exposure and minimizes the attack surface. As the number of entry points increases including edge devices, cloud endpoints, and communication protocols so does the attack surface. Security experts recommend adopting a zero-trust security model in which every telemetry packet is authenticated and encrypted, both at rest and in transit (Martis, 2025). Techniques such as mutual transport layer security, device identity verification, and role-based access control ensure the secure operation of edge agents and cloud services.
- c. **Bandwidth Efficiency and Cost Savings:** Local data filtering and pre-aggregation reduce unnecessary data flow to the cloud, minimizing transmission costs (Martis, 2025). Companies address cost control by implementing smart filtering, adaptive sampling, and edge-side analytics. Siemens, for example, uses edge analytics in factories, reducing unplanned downtime via predictive maintenance and cutting network loads by up to 70% (Martis, 2025).
- d. **Operational Resilience:** Local autonomy keeps systems running amid network disruptions, while distributed compute resources provide scalability and flexibility (Martis, 2025).

Review of Similar Works

Chen et al., (2023) conducted a comprehensive review of real-time analytics concepts, architectures, and machine learning and artificial intelligence considerations, developing a logical analytics stack to support the description of key functionality and relationships between relevant components in real-time analytics solutions. The authors highlighted complex event processing technology that has influenced many recent data streaming solutions in the analytics stack and discussed integrating machine learning and artificial intelligence into real-time analytics solutions. Their work guided practitioners in selecting appropriate solutions for real-time analytics problems, including the application of emerging AI technologies, and presented real-life case studies in finance and health domains to demonstrate the use of the techniques and technologies discussed.

Kyrama and Gounaris (2026) provided a comprehensive analysis of complex event processing, highlighting the distinctions between Complex Event Processing and Stream Processing systems, including the types of data processed and the operators used, as well as the emphasis on pattern detection. The authors provided an overview of existing optimization techniques adopted by Stream Processing and Complex Event Processing systems and discussed critical requirements for Complex Event Processing systems to ensure result correctness, such as handling inconsistencies, out-of-order and duplicate data, effective state management, and security considerations. The work investigated the deployment of Complex Event Processing systems in edge settings to address latency, security, and privacy concerns, highlighting challenges specific to these systems, and concluded with a vision for edge-based Complex Event Processing engines identifying research gaps and offering guidance for further exploration.

Challa et al., (2026) provided a comprehensive survey of anytime

algorithms tailored for data analytics over large datasets while emphasizing their application in time-sensitive decision-making environments. The authors examined the algorithmic foundations and state-of-the-art anytime approaches across various data analytics tasks, including classification, clustering, and frequent itemset mining. Qualitative analysis was presented for each algorithm based on key aspects such as interruptibility, resource adaptiveness, and solution quality under constrained conditions. The survey highlighted latest advancements and emerging research trends, providing insights into how anytime algorithms can be further developed to meet the demands of complex and dynamic environments.

Shaik, (2024) presented a comprehensive comparative analysis of three leading stream processing platforms: Apache Flink, Spark Streaming, and Kafka Streams, examining their architectural approaches, performance characteristics, and operational considerations in real-time data processing scenarios. Through extensive benchmarking and evaluation, the study investigated these platforms across multiple dimensions, including processing latency, throughput capacity, resource utilization, and operational complexity. The article revealed that Apache Flink demonstrates superior performance in low-latency scenarios with its true streaming model, while Spark Streaming excels in high-throughput situations with its micro-batch approach and robust ecosystem integration. Kafka Streams emerges as a compelling solution for lightweight stream processing needs, particularly in Kafka-centric architectures. Performance benchmarks indicated that Flink consistently achieves sub-100-millisecond latency for complex operations, Spark Streaming offers unparalleled throughput for large-scale data processing, and Kafka Streams provides the most straightforward operational model.

Abbasi et al., (2026) presented the first systematic comparative evaluation of Faust and Streamz, two Python-native Complex Event Processing engines representing fundamentally different architectural philosophies, specifically in the context of IoT network intrusion detection. Faust was selected for its actor-based stateful processing model with native Kafka integration and distributed table support, while Streamz was selected for its reactive, lightweight pipeline design targeting high-throughput stateless processing. The primary evaluation employed an IoT network intrusion dataset comprising 583,485 events from 83 heterogeneous devices, with a secondary IoT-adjacent benchmark using the PaySim financial transaction dataset of 6.4 million records. Results demonstrated that Streamz achieves superior throughput at 4450 events per second with 89% efficiency and minimal resource consumption of 40 MB memory and 12 ms median latency, while Faust provides robust intrusion pattern detection with 93% to 98% accuracy and stable, predictable resource utilization of 1.4% CPU standard deviation. Critical scalability thresholds were identified, with Streamz maintaining efficiency above 95% up to 3500 events per second while Faust degrades beyond 2500 events per second.

Martis, (2025) examined the integration of edge computing with cloud telemetry, presenting the latency-bandwidth-compute "design triangle" that helps engineers determine where processing occurs, what data gets prioritized, and how resilient the entire system is under real-world conditions. The article explored hybrid approaches that combine the benefits of cloud architectures with

edge computing, pushing compute closer to the data source and enabling faster decision-making for use cases like device health monitoring and predictive maintenance. Real-world applications across industries including industrial IoT, autonomous vehicles, smart cities, and healthcare were presented, with quantifiable benefits such as Siemens reducing network loads by up to 70%, Barcelona reducing congestion by 20%, and BIGO reducing hardware costs by 50% through adoption of modern streaming platforms (Martis, 2025).

Critical Comparative Analysis/ Synthesis of Prior Research
The reviewed techniques should not be considered competitors in

every context. Instead, they address different analytical requirements. Query-based systems are most appropriate when streaming analytics can be expressed through declarative operations. Rule-based systems are valuable when decisions must be transparent and deterministic. Stream-processing platforms provide the scalable computational infrastructure required to process continuous data. CEP adds sophisticated pattern recognition. Anytime algorithms address uncertain time and computational budgets. ML/AI introduces predictive intelligence, while edge-cloud architectures determine where computation should occur.

Table 6: Comparative Framework

Technique	Latency	Analytical Capability	Interpretability	Scalability	Resource Demand	Best Application
Query-based	Low–moderate	Aggregation	High	High	Moderate	Monitoring
Rule-based	Very low	Deterministic	Very high	Moderate	Low	Automated decisions
Stream processing	Low	General transformation	Moderate	Very high	Moderate–high	Large streams
CEP	Low	Complex patterns	High	Moderate	High	Fraud/security
Anytime	Variable	Progressive approximation	Depends on model	High	Adaptive	Time-critical decisions
ML/AI	Low–high	Predictive	Variable	High	High	Prediction
Edge–cloud	Application–dependent	Distributed	Variable	Very high	Distributed	IoT and cyber-physical systems

The key finding is that there is no universally optimal technique. Selection should depend on application latency requirements, data complexity, computational resources, required interpretability, scalability, security, energy constraints, and acceptable cost.

Evolution of Real-Time Big Data Analytics (2006-2026)

The evolution of real-time big data analytics reveals a clear trajectory from batch-oriented systems to stream-native architectures. This evolution has been documented in comprehensive surveys of the field (Chen *et al.*, 2023; Özçelik & Turhan, 2026).

Table 7: Evolution of Real-Time Big Data Analytics Techniques (2006-2026)

Period	Dominant Paradigm	Key Technologies	Latency Range	Key References
2006-2010	Batch Processing	Hadoop MapReduce	Minutes to hours	Chen <i>et al.</i> (2023)
2011-2013	Early Stream Processing	Apache Storm, Apache S4	Seconds to minutes	Cossa (2025)
2014-2016	Micro-Batch & Unified	Apache Spark Streaming	100ms to seconds	Shaik (2024)
2017-2019	True Stream Processing	Apache Flink, Kafka Streams	Milliseconds	Shaik (2024)
2020-2022	ML Integration	TensorFlow Extended, Apache Beam	Milliseconds	Chen <i>et al.</i> (2023)
2023-2026	Edge-Cloud Convergence	Hybrid architectures, Edge CEP	Sub-millisecond	Kyrama & Gounaris (2025)

Key Developmental Patterns

Pattern 1: From Batch to Stream-Native Architectures

Early systems like Hadoop MapReduce (2006) were designed for batch processing and could not meet the latency requirements of real-time applications. This limitation led to the development of stream-processing frameworks like Apache Storm (2011) and Apache Flink (2014), designed from the ground up for continuous data processing. As noted in comprehensive surveys of real-time analytics, the field has evolved significantly to address the velocity aspect of big data (Chen et al., 2023).

Pattern 2: Convergence of Batch and Stream Processing

Apache Spark (2014) introduced the concept of micro-batching, which treats streaming data as a series of small batches. This approach provides a unified programming model for both batch and stream processing, simplifying development and reducing operational complexity. Recent systematic reviews confirm that this convergence has become a dominant trend in the field (Özçelik & Turhan, 2026).

Pattern 3: Integration of Machine Learning with Real-Time Analytics

The integration of machine learning into stream processing systems has significantly enhanced analytical capabilities within real-time data environments. As documented by Chen et al. (2023), this convergence enables continuous deployment of machine learning models to evaluate and interpret incoming data streams, essential for applications such as predictive maintenance and anomaly detection.

Pattern 4: Edge-Cloud Convergence

The latency, bandwidth, and compute constraints form the "design triangle" that helps engineers determine where processing occurs. Kyrama and Gounaris (2025) provide a comprehensive analysis of edge deployment considerations for complex event processing systems and identify critical research gaps.

Comparative Analysis of Techniques

Table 8 presents a comprehensive comparison of all reviewed techniques across key performance dimensions, synthesized from the reviewed literature (Chen et al., 2023; Cossa, 2025; Shaik, 2024).

Table 8: Comprehensive Comparison of Real-Time Analytics Techniques

Technique	Latency	Throughput	Scalability	Fault Tolerance	Complexity	Best For
Query-based	Low-moderate	Moderate	High	Moderate	Low	Monitoring dashboards
Rule-based	Very low	Low	Moderate	Low	Low	Automated decisions
Stream Processing	Low	Very high	Very high	High	Moderate-high	Large-scale streams
CEP	Low	Moderate	Moderate	High	High	Pattern detection
Anytime Algorithms	Variable	Moderate	High	Moderate	Depends on model	Time-critical decisions
ML/AI Integration	Low-high	High	High	Variable	High	Predictive analytics
Edge-Cloud	Application-dependent	Distributed	Very high	High	Very high	IoT, cyber-physical systems

Research Gaps and Future Directions

Despite significant advancements, several research gaps remain, as identified in recent surveys (Challa et al., 2026; Kyrama & Gounaris, 2025):

- Standardized Benchmarking:** There is a lack of standardized benchmarks for comparing real-time analytics frameworks. Most studies use different datasets, metrics, and configurations, making direct comparison difficult (Shaik, 2024).
- Energy Efficiency:** The energy efficiency of real-time analytics systems has received limited attention. As edge devices proliferate, energy-efficient architectures become increasingly important (Kyrama & Gounaris, 2025).
- Edge-Based CEP:** Several advanced CEP solutions cannot be transferred to edge environments due to assumptions about abundant homogeneous resources and privacy constraints. Kyrama and Gounaris (2025) highlight this as a critical research direction.
- Federated Learning Integration:** Hybrid modular architectures integrating distributed processing,

machine learning, and NoSQL storage address varying requirements across finance, healthcare, and smart energy domains (Chen et al., 2023).

- Self-Optimizing Systems:** Self-optimizing systems that can automatically adjust processing parameters based on observed conditions represent a significant research direction (Cossa, 2025).

Challenges and Future Directions

The following presents the challenges and directions faced by real-time analytics systems.

Current Challenges

Despite significant progress, real-time data analytics systems face persistent challenges. Traditional databases and batch processing are still widely used today, while stream processing has been researched and proposed for many years. Systems differ across many aspects, including architecture, data models, languages, and processing mechanisms (Kyrama & Gounaris, 2026). The following are the challenges:

- a. Data Quality at Speed: Errors that batch pipelines catch during scheduled validation propagate instantly in streaming systems, requiring quality controls embedded in the ingestion layer itself (Chen et al., 2023). Streaming systems must produce accurate results when receiving out-of-order and delayed data because they lack control over input order (Kyrama & Gounaris, 2026).
- b. State Management: Effective state management in its broader sense is necessitated by the long-running nature of streaming queries, including parallelization, elasticity, load-shedding issues, and fault tolerance mechanisms (Kyrama & Gounaris, 2026). State management remains one of the most complex aspects of stream processing, particularly for stateful operations that maintain context across time and events.
- c. Security and Privacy: Security and privacy are closely related to where data is processed. Several advanced Complex Event Processing solutions cannot be transferred to an edge environment because they assume abundant, homogeneous resources; assume out-of-order and missing data are managed elsewhere; and assume privacy constraints on data sharing (Kyrama & Gounaris, 2026). Security experts recommend adopting a zero-trust security model with mutual transport layer security, device identity verification, and role-based access control (Martis, 2025).
- d. Cost Management: The infrastructure for real-time processing can become costly without optimized resource allocation. Transmitting massive data streams to the cloud can quickly become expensive, requiring smart filtering, adaptive sampling, and edge-side analytics to reduce unnecessary data flow (TMA Solutions, 2025).

Future Research Directions

The following are the future directions and areas of research in this study:

- a. AI-Driven Edge Analytics: Deploying more sophisticated AI models directly on edge devices represents a significant research direction (Martis, 2025). Modern systems leverage deep learning, reinforcement learning, and transformer models to enhance processing capabilities and enable sophisticated predictive insights.
- b. Self-Optimizing Systems: Self-optimizing systems that can automatically adjust processing parameters based on observed conditions without human intervention represent a significant advancement, including adaptive stream processing and edge computing integration (Martis, 2025).
- c. Edge-Native Platforms: Designing platforms specifically for distributed real-time processing rather than adapting cloud-native solutions represents a paradigm shift in the approach to real-time analytics.
- d. Energy-Efficient Architectures: Priority in this space is given to systems focusing on tight, high-efficiency scaling with low latency and high throughput.
- e. Federated Learning Integration: Hybrid modular architectures integrating distributed processing,

machine learning, and NoSQL storage address varying requirements across finance, healthcare, and smart energy domains.

- f. Edge-Based CEP Engines: Deploying CEP systems in edge settings addresses latency, security, and privacy concerns while highlighting challenges specific to these systems. Several advanced CEP solutions cannot be transferred to an edge environment because they rely on assumptions such as abundant homogeneous resources, out-of-order and missing data managed elsewhere, and privacy constraints on data sharing (Kyrama & Gounaris, 2026). Future research should address the challenges and limitations discovered in edge-based CEP deployment.

Conclusion

Real-time data analytics has evolved from a niche capability to a strategic necessity across industries. The primary techniques query-based analytics, rule-based analytics, programmatic platforms, Complex Event Processing, and anytime algorithms each offer distinct advantages and face unique limitations. Stream processing frameworks including Apache Flink, Apache Kafka, and Apache Storm provide the foundation for high-throughput, low-latency data processing. At the same time, Complex Event Processing enables sophisticated pattern detection for applications requiring complex event correlation. Integrating machine learning into streaming pipelines enhances analytical capabilities, enabling predictive insights and adaptive models. Edge-based deployment addresses latency, security, and privacy concerns but introduces challenges related to resource constraints and deployment complexity. The latency-bandwidth-compute design triangle provides a framework for understanding the trade-offs inherent in edge-cloud telemetry systems (Martis, 2025). Anytime algorithms provide flexible solutions for time-sensitive environments where computational resources or time availability are uncertain, with applications ranging from stock market analysis to fraud detection and path planning in robotics. The future of real-time data analytics lies in the convergence of classical and emerging paradigms, including AI-driven analytics, self-optimizing systems, and edge-native platforms. Success hinges on robust orchestration, data governance, and transforming multidisciplinary talent. Companies that balance the benefits, trade-offs, and unique business requirements of real-time analytics integration achieve resilient, intelligent systems that respond faster and deliver better insights over time. The exponential growth of heterogeneous data sources has driven a paradigm shift in how data is collected, processed, and used, and those who master this shift will define the next generation of business and technology innovation.

REFERENCES

- Abashes, G. M., Elshqirat, B. A. S. I. M. A., Abu-Shareha, A. A., & Alshraideh, M. O. H. A. M. M. A. D. (2025). Beyond the noise: unveiling meaningful patterns in mobile payment data using fuzzy logic and SVM. *J. Theor. Appl. Inf. Technol.*, 103(1), 266–284.
- Abbasi, M., Cardoso, F., Váz, P., Silva, J., Sá, F., & Martins, P. (2026). Performance comparison of Python-based complex event processing engines for IoT intrusion detection: Faust versus Streamz. *Computers*, 15(3), 200. <https://doi.org/10.3390/computers15030200>
- Al Khaldy, M., & Hamarsheh, A. (2026). From Data to Decisions:

- Implementing Real-Time Analytics Pipelines in Business Intelligence Architecture. In *Driving Modern Business Intelligence Architecture for Operational Efficiency* (pp. 137–166). IGI Global Scientific Publishing.
- Alam, P. K. (2025). Optimizing Apache Spark Structured Streaming Performance for External Data Enrichment: Balancing Latency and Throughput in Distributed Stream Processing. *Journal Of Engineering And Computer Sciences*, 4(8), 821–826.
- Asadova, F., Várkonyiné Kóczy, A. R., Kertész, G., & Lovas, R. (2023). A survey of the use of the anytime algorithm in fault detection in cloud systems. In *2023 IEEE 21st World Symposium on Applied Machine Intelligence and Informatics (SAMI)* (pp. 69–74). IEEE. <https://doi.org/10.1109/SAMI58000.2023.10044521>
- Babadzhanov, A., Yarashov, I., Juraev, M., Otakhonov, A., Kudaybergenov, A., & Utemuratov, R. (2025). Mathematical Representation of Netflow Analysis Decision Making Based on Production Logic. *International Journal of Advanced Computer Science and Applications*, 16(9).
- Baccouri, S., & Abdellatif, T. (2025). Big-abac: Leveraging big data for adaptive, scalable, and context-aware access control. *CMES-Computer Modeling in Engineering and Sciences*, 143(1), 1071–1093.
- Bhattacharyya, S., Banik, S., Paswan, A., Kumar, N., Bhar, A., Maiti, S., ... & Mondal, K. C. (2025, July). Comparative Analysis of Apache Spark and Apache Kafka for Efficient Data Integration for Real-Time Data Processing. In *International Conference on Computational Intelligence in Communications and Business Analytics* (pp. 59–72). Cham: Springer Nature Switzerland.
- Bolgov, S. (2025). Using Kafka in financial data processing systems: benefits and challenges. *Universum: technical sciences: electron. scientific journal*, 2(131), 13.
- Challa, J. S., Aarti, G. N., & Goyal, P. (2026). Time-sensitive data analytics: A survey of anytime techniques, applications and challenges. *Computer Science Review*, 59, 100850. <https://doi.org/10.1016/j.cosrev.2025.100850>
- Chandra, R., Agarwal, S., Kumar, S. S., & Singh, N. (2025). OCEP: an ontology-based complex event processing framework for healthcare decision support in big data analytics. *arXiv preprint arXiv:2503.21453*.
- Chen, R., Gu, B., & Ye, Z. (2025). Design and implementation of big data-driven business intelligence analytics system. In *Image Processing, Electronics and Computers* (pp. 1219-1227). IOS Press.
- Chen, W., Milosevic, Z., Rabhi, F. A., & Berry, A. (2023). Real-time analytics: Concepts, architectures, and ML/AI considerations. *IEEE Access*, 11, 71634–71657. <https://doi.org/10.1109/ACCESS.2023.3295694>
- Cossa, R. (2025). Real-time analytics in decision-making: Streaming data platforms and use cases. *Zenodo*.
- Das, S., & Rajput, G. K. (2026). Performance Analysis of Rule-Based Reasoning in Complex Data Environments. In *Intelligent Technologies for Scientific Research and Engineering* (pp. 86-103). Bentham Science Publishers.
- El-Kafrawy, P., & El-Amin, M. F. (2026). An overview of big data analytics. *Mathematical Modeling for Big Data Analytics*, 3–18.
- Goswami, A., & Shivaji, G. B. (2025, February). Using big data and Kafka to track resource utilization in real time. In *2025 International Conference on Intelligent Control, Computing and Communications (IC3)* (pp. 1028–1034). IEEE.
- Gudduri, G. (2026). *Enhancing IoT Data-Stream Processing with a Lean Serverless Cloud Architecture* (Doctoral dissertation, Dublin, National College of Ireland).
- Haapala, H. (2026). Transition from Batch to Real-Time Analytics: Implications for Analytics Data Modeling.
- Harditya, M., Taufiqoh, S. D., Somadias, V., Priyatna, K. D., & Sari, R. F. (2026, June). Enhancing AI-Driven Query Generator by Bridging Natural Language and Cloud Database. In *19th International Conference on Quality in Research (QiR, 2025)* (pp. 154- 169). Atlantis Press.
- Hassan, N. A. B. (2025). Managing data dependencies in cloud-based big data pipelines: Challenges, solutions, and performance optimization strategies. *Orient Journal of Emerging Paradigms in Artificial Intelligence and Autonomous Systems*, 15(2), 20–28.
- Igulu, K. T., Wisdom, D. D., Arowolo, M. O., & Singh, T. P. (2025). Computational intelligence in big data analytics. In *Computational Intelligence in Industry 4.0 and 5.0 Applications* (pp. 265–328). Auerbach Publications.
- Immadisetty, A. (2025). *Data engineering with a focus on scalable platforms and real-time analytics*. Notion Press.
- Javed, N., & Rajumar, R. Y. (2026, April). Big Data Event Streaming with Apache Kafka for Improved Data Flows in IoT using Optimized Kafka-Based Data Streaming Workflow. In *Global Conference on Sustainable Energy Systems, Smart Electronics and Intelligent Computing (GCSESEIC 2025)* (pp. 277–287). Atlantis Press.
- Joseph, N. S. (2025, November). DIGITAL PLATFORMS AND ECOSYSTEMS: REDEFINING BUSINESS MODELS IN THE AGE OF CONNECTIVITY. In *AUN INTERNATIONAL CONFERENCE* (Vol. 3, No. 1).
- Khattach, O., Moussaoui, O., & Hassine, M. (2025). End-to-end architecture for real-time IoT analytics and predictive maintenance using stream processing and ML pipelines. *Sensors*, 25(9), 2945.
- Kumar, S. S., & Agarwal, S. (2024). Rule-based complex event processing for IoT applications: Review, classification and challenges. *Expert Systems*, 41(9), e13597.
- Kyrama, S., & Gounaris, A. (2025). Complex event processing: Current status and considerations for edge deployment: future *Generation Computer Systems*, 177, 108247.
- Kyrama, S., & Gounaris, A. (2026). Complex event processing: Current status and considerations for edge deployment. *Future Generation Computer Systems*, 177, 108247. <https://doi.org/10.1016/j.future.2025.108247>
- Langhi, S., Tommasini, R., Bonifati, A., & Bernhardt, T. (2025, June). EPL: The event processing language for streaming data. In *Proceedings of the 19th ACM International Conference on Distributed and Event-based Systems* (pp. 62–73).
- Li, Y. (2026). Research on the Optimization of Enterprise Management Information System Driven by Big Data—*Journal of Technology Innovation and Engineering*, 2(4).

- Lin, F., Tang, B., Qin, G., Sun, S., & Lin, J. (2025, June). Real-Time Analysis Model of Teaching Quality for Higher Education Based on Complex Event Processing, in *Proceedings of the 2nd International Conference on Intelligent Education and Computer Technology* (pp. 421-426).
- Liyang, S. (2025). The Digital Media Era: Publishing and Communication Ecosystem Undergoes Metamorphosis, Content Innovation Sets Sail. *Journal of Sociology and Education*, 1(10).
- Manjusha, G., & Kumar Sharma, A. (2026). Real-Time Predictive Analytics on Streaming Data. Available at SSRN 6331458.
- Martis, I. (2025). *On the edge and in the cloud: The future of real-time data flows*. *Analytics Magazine*. <https://doi.org/10.1287/LYTX.2025.04.19>
- Medina-Bulo, I., Garcia-Dominguez, A., & Gutiérrez-Madroñal, L. (2025). Strong mutants and Subsuming mutants study for EPL language. *Strong mutants and Subsuming mutants study for EPL language*.
- Mehmood, A., Arif, M., & Mehmood, F. (2025). Towards a unified digital ecosystem: the role of platform technology convergence. *Electronics*, 14(24), 4787.
- Menon, P. K. (2025). ADAPTIVE STREAMING INTELLIGENCE FOR REAL-TIME FINANCIAL FRAUD DETECTION: A KAFKA-ORIENTED MACHINE LEARNING FRAMEWORK. *Ethiopian International Journal*, 12(11), 612–620.
- Modalavalasa, G. (2026). *Application of Data Engineer and Their Architecture*. Geh Press.
- Neoaz, N. (2025). Big Data Analytics: Study the implications of big data analytics on decision-making processes in organizations. *Author Nahid Neoaz*.
- Önden, A. (2026). A systemic approach to decision support and automation: the role of big data analytics and real-time processing in management information systems. *Systems*, 14(2), 216.
- Özçelik, T. O., & Turhan, S. N. (2026). IoT-generated real-time big data analytics: A systematic literature review. *DergiPark*.
- Parmar, T. (2025). Scaling data infrastructure for high-volume manufacturing: challenges and solutions in big data engineering. *International Scientific Journal of Engineering and Management*, 4(01), 10–55041.
- Patil, R. S., & Moantri, S. (2026). Challenges and Opportunities in Real-Time Data Processing: Advancements and Limitations in Real-Time Data Analytics. *Artificial Intelligence and Machine Learning in Neurology*, 2, 647–683.
- Poss, L., & Schöning, S. (2026). A synergistic engine paradigm for real-time, context-aware decision-making: integrating declarative processes and event streams. *Complex & Intelligent Systems*, 12(4), 136.
- Premalatha, N. (2026). Chapter 4: Big Data Processing Frameworks: Batch, Stream, and Hybrid Computing Models. *Big Data Foundations: Architectures, Analytics and Intelligent Decision Systems*, 61.
- Puslecki, T., & Walkowiak, K. (2025, September). Data Stream Processing for Resource-Constrained TinyML Systems. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (pp. 161–173). Cham: Springer Nature Switzerland.
- Qrvey. (2026). *Real-time analytics: How it works, benefits & challenges*
- Richards, M. E. (2025). ECA Rules in EmbedDB: Enabling Lightweight, Event-Driven Decision Making for Resource-Constrained Devices.
- Seidl, T. (2012). Stream data mining and anytime algorithms. *GFKL 2012 Abstract*.
- Shahnawaz, M., & Kumar, M. (2025). A comprehensive survey on big data analytics: Characteristics, tools and techniques. *ACM Computing Surveys*, 57(8), 1–33.
- Shaik, A. S. (2024). Advancements in real-time stream processing: A comparative study of Apache Flink, Spark Streaming, and Kafka Streams. *International Journal of Computer Engineering and Technology*, 15(6), 631–639. <https://doi.org/10.5281/zenodo.14216515>
- Shamsinejad, E., Baniroostam, T., Pedram, M. M., & Rahmani, A. M. (2025). A review of anonymization algorithms and methods in big data. *Annals of Data Science*, 12(1), 253–279.
- Sharma, V., Vashishth, T. K., Sharma, K. K., Chaudhary, S., Kumar, B., & Panwar, R. (2026). The Role of AI and Big Data Analytics in Smart Cities: Leveraging Digital Platforms, Cloud Computing, and IoT. *Digital Cities*, 351–376.
- Shermy, R. P., & Saranya, N. (2025). Cloud-based Big Data architecture and infrastructure. *Resilient Community Microgrids*, 131–188.
- Siddiqui, A. R., & Qureshi, U. A. (2025). A Systematic Evaluation of Modern Data Ingestion Architectures for Scalable and Real-Time Enterprise Analytics Pipelines. *International Journal of Multidisciplinary Science, Engineering and Mathematics*, 15(9), 1–19.
- Singh, S. (2026). Intelligent Data Analytics on Cloud Infrastructure: Real-Time Insights and Managed Service Models. *International Journal of Distributed Computing and Technology*, 12(1), 36–47.
- Yip, J., Gupta, N., & Wojtyczka, M. (2026). Real-Time Intelligence with Spark Structured Streaming. In *Databricks Data Intelligence Platform: Powering the Agentic Era* (pp. 263–286). Berkeley, CA: Apress.