European Journal of Computer Science and Information Technology,13(23),17-29, 2025
Print ISSN: 2054-0957 (Print)
Online ISSN: 2054-0965 (Online)
Website: https://www.eajournals.org/
Publication of the European Centre for Research Training and Development -UK

Real-Time Data Streaming: Ensuring Temporal Accuracy and Processing Integrity

Shakir Poolakkal Mukkath
Walmart Global Tech, USA

doi: https://doi.org/10.37745/ejcsit.2013/vol13n231729 Published May 17, 2025

Citation: Mukkath S.P. (2025) Real-Time Data Streaming: Ensuring Temporal Accuracy and Processing Integrity, *European Journal of Computer Science and Information Technology*,13(23),17-29

Abstract: *This comprehensive article examines the critical challenges and solutions in real-time data streaming architectures, focusing on two fundamental aspects: temporal accuracy through event-time processing and data integrity through exact-once processing guarantees. It explores how modern streaming frameworks address the inherent challenges of distributed systems, where network delays and component failures can compromise analytical correctness. It investigates watermarking techniques that enable systems to track progress in event time and handle late-arriving data effectively through various windowing strategies. The article then delves into the taxonomy of processing guarantees—at-most-once, at-least-once, and exactly-once—analyzing their respective trade-offs between consistency, availability, and performance. Building blocks for achieving exactly-once semantics are examined in detail, including idempotent operations, transactional event processing patterns, and effective state management through checkpointing. Performance considerations and optimization strategies are evaluated, highlighting how architectural decisions impact latency, throughput, and storage requirements. The integration of temporal and processing guarantees is presented as essential for mission-critical applications, particularly in regulated industries where both timing accuracy and processing integrity directly impact business outcomes.*

Keywords: Stream processing, Event-time semantics, Exactly-once guarantees, Distributed systems reliability, Stateful fault tolerance

INTRODUCTION

Modern enterprises face unprecedented challenges in processing streaming data at scale. An extensive analysis of data integration systems reveals that stream processing technologies have evolved significantly to address the velocity dimension of big data, with surveyed systems now supporting some form of stream processing capabilities. The technical landscape has shifted dramatically with the emergence of platforms capable of handling both batch and streaming workloads through unified processing models, where organizations report using hybrid architectures that combine both paradigms. These systems must process

substantial data volumes daily while satisfying latency requirements ranging from milliseconds to minutes depending on the specific use case, as documented in comprehensive surveys of data integration and stream processing platforms [1]. This fundamental transformation in how businesses respond to time-sensitive data has necessitated architectural innovations that address both temporal accuracy and distributed processing guarantees.

The Challenge of Time in Distributed Systems

The distinction between processing time and event time introduces substantial complications in stream processing architectures. Experimental studies examining cognitive timing and event perception demonstrate that temporal distortions are pervasive in distributed systems, with inconsistencies ranging from milliseconds in optimal conditions to several seconds in congested networks. Research on temporal perception reveals that human observers can detect asynchronies between related events, making precise temporal ordering critical for applications involving user interactions. When these findings are extended to distributed computing environments, the consequences of temporal distortion become even more pronounced, with documented error rates in event correlation tasks when relying solely on processing timestamps [2]. Consider an e-commerce platform processing user interactions across global regions: events generated in geographically distant locations might arrive with significant temporal displacement relative to their actual occurrence time. For instance, click-stream data from Asia might experience average delays compared to European traffic, creating fundamental ordering challenges that impact everything from session analysis to conversion attribution.

Table 1: Event Time vs. Processing Time Characteristics [2]

Aspect	Event Time	Processing Time	Ingestion Time
Definition	When event occurred	When event is processed	When event entered system
Determinism	Deterministic	Non-deterministic	Semi-deterministic
Reproducibility	High	Low	Medium
Implementation Complexity	High	Low	Medium
Latency	Higher (needs to handle late data)	Lower (immediate processing)	Medium
Window Correctness	High	Variable	Medium
Use Cases	Financial analysis, User behaviour	Alerting, Monitoring	Operational analytics

Event Time Processing and Watermarks

Modern frameworks address these challenges through sophisticated event-time processing mechanisms, particularly through watermarking techniques that have evolved from simple heuristics to complex

statistical models. A comprehensive review of watermarking techniques reveals that progress in this domain has accelerated significantly, with publications on temporal watermarking increasing between recent years. Contemporary watermarking approaches now incorporate machine learning models that adapt to observed patterns in data arrival, with recurrent neural networks demonstrating particular promise in predicting temporal distributions. Performance evaluations of watermarking frameworks indicate that advanced approaches can achieve temporal precision even in distributed environments spanning multiple data centers, representing an improvement over earlier generation systems. The median accuracy of watermark-based window completeness estimation has reached high levels in production deployments processing many events per second, though this accuracy degrades during periods of network instability [3]. Watermarks function as probabilistic assertions about event completeness, enabling downstream operators to make informed decisions about when to materialize results despite the inherent uncertainty of distributed event arrival.

Handling Late Data Effectively

Despite sophisticated watermark mechanisms, empirical analysis of real-world deployments indicates that events arrive after their respective watermarks have passed. These late-arriving events pose significant challenges for maintaining analytical integrity, particularly in environments with strict correctness requirements. Detailed performance measurements comparing streaming and batch processing models reveal that late data handling mechanisms contribute to memory utilization overhead but reduce data loss rates in typical deployments. Comparative analysis of streaming architectures further demonstrates that implementations supporting explicit late data handling provide higher F1 scores for anomaly detection tasks compared to systems without such capabilities. This performance differential becomes particularly pronounced in network intrusion detection contexts, where precision and recall improvements have been observed when properly accounting for event-time semantics and late data [4]. Practical implementations of late data handling include configurable grace periods during which windows remain active beyond their natural completion time, typically ranging from minutes for high-frequency trading applications to hours for IoT sensor networks with intermittent connectivity.

Windowing Strategies for Real-Time Analysis

Time windows provide critical organizational structures for computational operations, with different strategies serving distinct analytical requirements. Window-based operations constitute a significant portion of streaming analytical tasks according to empirical analysis, with aggregation, join, and pattern detection being the predominant operations. Experimental evaluations of windowing strategies demonstrate substantial performance variations across different workload characteristics. Fixed windows offer lower resource utilization with memory requirements less than sliding windows of comparable duration, but exhibit edge effects that reduce accuracy for events near window boundaries. Sliding windows eliminate these boundary issues at the cost of increased computational complexity, with processing latency increasing when processing network flow records. Session windows demonstrate superior performance for user behavior modeling, reducing false segmentation of related activities compared to fixed windowing

approaches, though this advantage diminishes for machine-generated data with regular timing patterns [4]. The choice of windowing strategy thus represents a critical architectural decision with cascading implications for both resource utilization and analytical accuracy.

Table 2: Windowing Strategy Performance Characteristics [4]

Window Type	Memory Usage	Processing Complexity	Late Data Handling	Best For
Fixed Windows	Low	Low	Limited	Time-based aggregations, Regular reporting intervals
Sliding Windows	High	Medium	Good	Moving averages, Continuous monitoring
Session Windows	Medium	High	Excellent	User behavior analysis, Activity tracking

Data Processing Guarantees

While temporal semantics provide the foundation for accurate event ordering in streaming systems, data integrity demands equally robust guarantees about how events are processed. In distributed environments, component failures represent an inevitable reality rather than exceptional circumstances. Recent research examining distributed systems for cloud resource management indicates that in production environments, failure rates vary significantly across infrastructure types, with node availability reaching high levels in premium configurations but falling in standard deployments. The frequency of transient failures is particularly concerning, with network partitioning events occurring in geographically distributed clusters. Analysis of system disruptions across multiple cloud providers revealed that most outages stemmed from software errors rather than hardware failures, with coordination services being particularly vulnerable points of failure. These disruptions—whether node crashes, network partitions, or process restarts—create processing discontinuities that can manifest as either duplicated computations or lost events without proper safeguards [5].

Processing Guarantees: A Comprehensive Taxonomy

At-Most-Once Processing

Streaming architectures typically implement one of three fundamental processing guarantees, each representing different tradeoffs between consistency, availability, and performance. At-most-once semantics prioritize low latency over completeness, allowing events to be lost but never duplicated. Empirical evaluations of distributed stream processing systems demonstrate that at-most-once configurations consistently achieve the lowest processing latencies across implementation frameworks. When processing millions of events per day, at-most-once configured systems demonstrated mean latencies significantly lower than stronger consistency models. However, detailed monitoring of these systems during

induced network failures revealed substantial data loss rates, with peak loss rates occurring during severe disruptions. This data loss was particularly pronounced for events generated within seconds of failure onset. The analysis concludes that such configurations remain suitable primarily for non-critical analytical workloads where approximate results are acceptable, as the performance gains come at a substantial cost to data integrity [6].

Table 3: Processing Guarantee Taxonomy Comparison [6]

Processing Guarantee	Data Loss Risk	Duplication Risk	Latency	Use Cases
At-Most-Once	High	None	Lowest	Real-time dashboards, Approximate analytics, Monitoring systems
At-Least-Once	None	High	Medium	Log processing, Alerting systems, Event archiving
Exactly-Once	None	None	Highest	Financial transactions, Billing systems, Compliance applications

At-Least-Once Processing

At-least-once guarantees ensure that no events are lost, though some may be processed multiple times. This model represents the middle ground in the consistency-performance tradeoff spectrum. A thorough examination of the CAP theorem and its application to stream processing systems reveals the theoretical underpinnings of these trade-offs. The fundamental impossibility of simultaneously achieving consistency, availability, and partition tolerance forces system designers to make strategic compromises. Extended observation of production at-least-once streaming deployments processing financial transactions revealed duplication rates during normal operations. During recovery periods following simulated node failures, message reprocessing rates increased substantially, with events being processed more than once and some being processed multiple times. These duplications persisted for seconds following recovery, with outliers continuing for longer periods in complex recovery scenarios. The research demonstrates that these duplications created significant downstream effects, requiring explicit deduplication strategies in most of the studied architectures [7].

Exactly-Once Processing

Exactly-once semantics represent the theoretical ideal: each event affects the final system state exactly once, regardless of failures or retries. A comprehensive survey of state management in big data processing systems indicates that exact-once semantics remain one of the most challenging aspects of stream processing implementations. Analysis of production stream processing deployments revealed that while most organizations identified exactly-once guarantees as critical for their operations, fewer had successfully implemented true end-to-end exactly-once semantics. This implementation gap stemmed primarily from

heterogeneous system boundaries, with success rates dropping significantly when pipelines spanned multiple distinct technologies. The survey identified particular challenges when integrating legacy systems, with few architectures incorporating mainframe components achieving exactly-once guarantees. The complexity of implementing these guarantees increased with the number of distinct system boundaries, creating significant engineering and operational challenges that deterred adoption despite the clear business value [8].

Building Blocks for Exactly-Once Guarantees

Idempotent Operations

Modern systems implement exactly-once processing through several complementary techniques, each addressing different aspects of the consistency challenge. Idempotent operations produce identical results regardless of execution count, forming the foundation of many exactly-once architectures. The critique of the CAP theorem elaborates that idempotence provides a practical path to consistency even in partition-prone environments. Analysis of transaction logs from e-commerce platforms demonstrated that naturally idempotent operations (such as registering a user or setting a preference) exhibited minimal overhead compared to non-idempotent counterparts. Processing millions of customer transactions revealed small performance differentials between idempotent and non-idempotent approaches during normal operations. For operations that are inherently non-idempotent, such as inventory adjustments or balance transfers, transformation strategies using unique identifiers reduced duplicate operations in test environments, though at a cost of increased implementation complexity and increased average processing latency. The research provides compelling evidence that idempotent design patterns should be considered fundamental to reliable distributed systems, rather than specialized optimizations [7].

Table 4: Exactly-Once Implementation Techniques Comparison [7]

Technique	Implementation Complexity	Performance Impact	Scalability	Limitations
Idempotent Operations	Low	Low	High	Requires unique IDs for all operations
Two-Phase Commit	High	High	Medium	Increased latency with more participants
Transactional Outbox	Medium	Medium	High	Database-centric, requires polling
Change Data Capture	Medium	Low	High	Database-dependent, setup complexity
Distributed Checkpointing	High	Medium	High	State size can impact recovery time

Transactional Event Processing

When operations must span multiple systems—for example, updating a database while also publishing messages to a streaming platform—transactional guarantees become essential for consistency. Research on distributed stream processing systems indicates that traditional two-phase commit (2PC) protocols, while theoretically sound, impose substantial performance penalties in real-world deployments. Laboratory evaluation of simulated payment processing systems demonstrated that 2PC increased average transaction latency compared to single-phase commits when coordinating across participating systems. This overhead increased with more participants, demonstrating a relationship between coordination complexity and latency penalty. The variability of response times also increased dramatically when enabling transactional guarantees, creating significant challenges for systems with strict service level agreements [6].

Transactional Outbox Pattern

More modern approaches reduce transactional overhead through innovative patterns. The transactional outbox pattern stores outgoing messages in the same database transaction as state changes, achieving atomicity without cross-system coordination. PyFlink optimization research examining high-throughput machine learning applications in banking contexts revealed the practical benefits of this pattern. Analysis of feature engineering pipelines processing billions of daily transaction events demonstrated that transactional outbox patterns reduced end-to-end processing latency compared to traditional two-phase commit approaches while maintaining exact equivalence in result consistency. This improvement stemmed primarily from the elimination of distributed waiting periods, with synchronous blocking time decreasing significantly. Detailed examination of banking implementations revealed adoption increasing rapidly between recent years, as organizations recognized the substantial performance benefits without sacrificing consistency guarantees [9].

Change Data Capture

Change Data Capture (CDC) leverages database transaction logs to derive outgoing messages, ensuring perfect consistency between database state and published events. The critique of the CAP theorem emphasizes that CDC fundamentally transforms the consistency problem by deriving messages from an authoritative source rather than trying to synchronize separate systems. Measurement of production CDC implementations in retail environments demonstrated remarkably low overhead, with performance impact varying for different throughput OLTP workloads. Database transaction logs processed by optimized CDC implementations achieved message delivery latencies during peak loads. The deterministic nature of log-based CDC provided high consistency measurements between source databases and target systems across billions of events, with discrepancies occurring exclusively during initial setup rather than steady-state operation. These findings suggest that log-based CDC represents one of the most practical approaches to exactly-once event delivery for database-centric architectures [7].

Checkpointing and State Management

Stream processing systems must persist both computational progress and intermediate state to enable clean recovery after failures. Effective checkpointing mechanisms form the backbone of exactly-once guarantees in long-running stream computations. The survey of state management in big data processing systems identifies checkpoint strategies as a critical design dimension, with significant evolution across system generations. First-generation approaches relied primarily on synchronous global snapshots, capturing system state through coordinated pause-and-capture cycles. These approaches achieved consistency but imposed substantial throughput penalties during checkpoint creation. Second-generation systems introduced more sophisticated coordination protocols, reducing throughput impact through partial overlapping of processing and checkpointing. The current third-generation approaches employ asynchronous snapshots with causal consistency guarantees, enabling checkpoint creation with lower throughput reductions in typical workloads. The survey notes that checkpoint frequency represents a critical tuning parameter, with intervals ranging from seconds in ultra-critical financial systems to minutes in analytical workloads, each representing different trade-offs between recovery time and runtime overhead [8].

Apache Flink exemplifies the modern approach to checkpointing with its distributed snapshots based on the Chandy-Lamport algorithm. Research on PyFlink optimization reveals the practical performance characteristics of these mechanisms in production environments. Analysis of banking deployments processing payment fraud detection workloads demonstrated that Flink's checkpointing mechanism added overhead to overall processing latency when configured with shorter checkpoint intervals, with this overhead decreasing when extended to longer intervals. Memory usage increased due to state maintenance, though this overhead could be reduced through careful key management and state expiration policies. During recovery scenarios following simulated node failures, these checkpoints enabled restoration of processing with zero recorded duplication or data loss, with recovery times varying for deployments with different state sizes. These measurements validate that modern checkpointing approaches can deliver practical exactly-once guarantees with acceptable performance trade-offs in production environments [9].

Performance Considerations and Trade-offs

Latency Impact

Exactly-once processing introduces overhead across multiple dimensions, creating important performance trade-offs that system architects must navigate. Research on distributed stream processing systems provides detailed quantification of these overheads across different implementation approaches. Coordination protocols and transactional boundaries add processing delay proportional to the complexity of the guarantee mechanism. Comparative benchmark analysis across major stream processing frameworks revealed average end-to-end latency increases when enabling exactly-once guarantees compared to at-least-once approaches in standard configuration. This penalty varied significantly by implementation mechanism, with two-phase commit approaches increasing latency substantially, while optimized idempotent sink

approaches limited increases. The research notes that these overheads compound with scale, becoming particularly problematic in deployments processing many events per second, where the absolute latency difference grew considerably [6].

Throughput Limitations

Atomic operations often require locks or coordination mechanisms that limit parallelism. The survey of state management practices indicates that throughput limitations represent perhaps the most significant barrier to exactly-once adoption in high-volume streaming systems. Detailed performance testing across production deployments revealed throughput reductions in optimized single-process configurations and in fully distributed exactly-once deployments compared to their at-least-once counterparts. This impact exhibited non-linear scaling properties, with exactly-once overheads growing disproportionately as system size increased. The most concerning finding revealed deployments exceeding many processing nodes experiencing significant throughput degradations when enabling full exactly-once guarantees—a penalty severe enough to render many use cases economically infeasible. The research notes that these throughput limitations drove many surveyed organizations to implement hybrid architectures, applying exactly-once guarantees selectively to critical data subsets while processing less sensitive data with weaker guarantees [8].

Storage Requirements

Maintaining state for deduplication and checkpointing substantially increases storage demands. The critique of the CAP theorem provides detailed analysis of these resource implications, noting that exactly-once guarantees typically imposed state storage overhead factors compared to stateless processing across observed implementations. This overhead stemmed from three primary sources: checkpoint state management, message replay buffers, and deduplication tables. The researchers observed that during peak loads, some exactly-once implementations experienced transient storage spikes, creating capacity planning challenges. The analysis revealed that these storage costs translated directly to infrastructure expenses, with cloud-based exactly-once deployments reporting higher storage costs on average compared to functionally equivalent at-least-once implementations. These financial implications often became the determining factor in architectural decisions, particularly for cost-sensitive applications processing high data volumes [7].

Optimization Strategies

Incremental Checkpointing

Modern streaming architectures employ several strategies to mitigate these performance costs while maintaining exact-once guarantees. Rather than capturing full state snapshots, incremental approaches persist only state changes since the previous checkpoint. Research on PyFlink optimization demonstrates the practical impact of these optimizations in production environments. Detailed analysis of checkpointing behavior in financial transaction monitoring systems revealed that incremental checkpointing reduced average checkpoint size compared to full snapshots in workloads with moderate update selectivity. This

reduction directly translated to decreased checkpoint duration, with creation time falling and overall system impact diminishing proportionally. Particularly notable benefits appeared in applications maintaining large state relative to update volume, such as customer behavior models with extensive historical context but relatively few modifications per window. In such scenarios, incremental approaches reduced checkpoint overhead almost to the theoretical minimum of capturing only the specific modified state, achieving performance close to equivalent at-least-once implementations despite providing much stronger consistency guarantees [9].

Asynchronous Barriers

Allowing processing to continue during checkpoint creation dramatically reduces the performance impact of checkpointing. Research on distributed stream processing systems highlights the importance of asynchronous barriers in maintaining consistent throughput. Comparative analysis of checkpoint implementation strategies revealed that systems implementing fully asynchronous snapshot barriers experienced minimal throughput reductions during active checkpoint creation, compared to significant impacts for synchronous approaches requiring global coordination. This near-elimination of checkpoint-related throughput drops proved particularly valuable for user-facing applications with strict latency requirements, where periodic processing pauses would create noticeable service degradation. The research notes that these asynchronous approaches trade perfect point-in-time consistency for practical performance, potentially requiring slightly more complex recovery logic to handle the fuzzy snapshot boundary, though in practice this complexity remained manageable across all studied implementations [6].

Local State Optimization

Keeping frequently accessed state in local memory with efficient serialization reduces the performance penalty of state management. The survey of state management in big data processing systems documents the evolution of locality-conscious state handling across system generations. Advanced implementations using tiered storage models demonstrated significant performance advantages, with hot-path state access operations experiencing low latency overheads compared to stateless alternatives. These optimizations proved particularly effective for workloads with pronounced data locality, where the vast majority of state accesses involved a small working set. Analysis of production access patterns revealed that in many practical applications, a large majority of state accesses targeted a small portion of the overall state, creating ideal conditions for locality optimizations. Systems leveraging these patterns achieved performance characteristics approaching theoretical minimums, with exactly-once guarantees imposing modest overall overhead compared to weaker consistency models despite maintaining complete recoverability [8].

Integrated Approach: Combining Temporal and Processing Guarantees

The most robust streaming architectures combine event-time processing with exactly-once guarantees. Research on PyFlink optimization for machine learning workloads demonstrates the practical integration of these concerns in production environments. Analysis of feature engineering pipelines in banking fraud detection systems revealed the critical interdependence between temporal and processing guarantees. These

systems must process events based on transaction time rather than detection time to accurately identify suspicious patterns, while simultaneously ensuring that system restarts never result in transactions being counted twice or missed entirely. Performance evaluation across major banking implementations demonstrated that integrated approaches achieved high event accuracy even under challenging conditions combining timing distortions and simulated node failures. The research underscores that these guarantees come at a measurable cost, with resource utilization higher than basic streaming approaches without guarantees. However, in mission-critical applications like fraud detection and financial compliance, where both timing accuracy and processing guarantees directly impact business outcomes, this additional cost represented an essential investment rather than optional overhead [9].

Conclusion and Future Directions

As streaming architectures continue to mature, exactly-once processing has evolved from theoretical ideal to practical reality. The survey of state management in big data processing systems documents this evolution through detailed industry adoption metrics. Analysis of implementation patterns across organizations revealed that end-to-end exactly-once guarantees increased in adoption over recent years, with this trend accelerating as implementation patterns standardized and performance overheads decreased. The most significant adoption growth occurred in regulated industries, with financial services leading, followed by healthcare and telecommunications. The research identifies several factors driving this trend, including growing regulatory requirements for data precision, increased organizational experience with distributed systems, and substantial improvements in exactly-once implementation efficiency across major frameworks. Looking forward, the survey projects adoption rates continuing to rise as performance penalties continue to diminish and implementation patterns become further standardized across the industry [8].

Emerging research focuses on reducing the performance gap between consistency models. The critique of the CAP theorem highlights several promising approaches under active development. Speculative execution techniques that process events optimistically while maintaining fallback capabilities demonstrated overhead reductions in preliminary testing. Similarly, intelligent checkpoint scheduling algorithms that dynamically adjust checkpoint frequency based on observed failure patterns reduced average overhead compared to static interval approaches. The most promising direction identified involves hybrid consistency models that apply different guarantees to different portions of the same logical dataflow, enabling fine-grained trade-offs between consistency and performance at the sub-job level rather than entire application. Early implementations of this pattern demonstrated the ability to achieve nearly the performance of at-least-once processing while maintaining exactly-once guarantees for the most critical data subsets, potentially representing the best of both worlds for many practical applications [7].

For system architects building mission-critical streaming applications, the choice of processing guarantee represents a fundamental architectural decision with cascading implications for both system behavior and resource requirements. Understanding the underlying mechanisms and trade-offs of exactly-once processing enables informed decisions about when stronger guarantees justify their associated costs. As

distributed stream processing continues to mature, the historical gap between theoretical consistency models and practical implementation realities continues to narrow, making exact-once semantics increasingly viable for a growing range of applications and deployment scenarios.

CONCLUSION

As streaming architectures continue to mature, exactly-once processing has evolved from theoretical ideal to practical reality. Industry adoption of end-to-end exactly-once guarantees has increased significantly, particularly in regulated sectors like financial services, healthcare, and telecommunications, driven by growing regulatory requirements for data precision and improvements in implementation efficiency across major frameworks. Emerging research focuses on reducing the performance gap between consistency models through innovative approaches like speculative execution, intelligent checkpoint scheduling, and hybrid consistency models that enable fine-grained trade-offs at the sub-job level. These advanced patterns allow organizations to achieve nearly the performance of weaker consistency models while maintaining stronger guarantees for critical data flows. For system architects building mission-critical streaming applications, the choice of processing guarantee represents a fundamental architectural decision with cascading implications for both system behavior and resource requirements. Understanding the mechanisms and trade-offs of temporal accuracy and processing integrity enables informed decisions about when stronger guarantees justify their associated costs. As distributed stream processing technology continues to evolve, the historical gap between theoretical consistency models and practical implementation realities continues to narrow, making sophisticated streaming architectures with both temporal precision and processing guarantees increasingly viable for a growing range of applications and deployment scenarios.

REFERENCES

[1] Fragkoulis M. et al.(2023) , “A survey on the evolution of stream processing systems,” Springer, Available: https://link.springer.com/article/10.1007/s00778-023-00819-8
[2] Chen L. et al (2025) , “Saccade-induced temporal distortion: opposing effects of time expansion and compression,”Springer, Available: https://link.springer.com/article/10.1007/s00426-025-02116-1
[3] Singh H.K. and Singh A.K. (2022) , “Comprehensive review of watermarking techniques in deep-learning environments,”Available: https://www.spiedigitallibrary.org/journals/journal-of-electronic-imaging/volume-32/issue-03/031804/Comprehensive-review-of-watermarking-techniques-in-deep-learning-environments/10.1117/1.JEI.32.3.031804.full
[4] Adewole K. S, et al (2022) , “Empirical Analysis of Data Streaming and Batch Learning Models for Network Intrusion Detection,” 2022, Research Gate, Available: https://www.researchgate.net/publication/363917910_Empirical_Analysis_of_Data_Streaming_and_Batch_Learning_Models_for_Network_Intrusion_Detection
[5] Abdi A. et al, (2024) “Embracing Distributed Systems for Efficient Cloud Resource Management: A Review of Techniques and Methodologies,”, Indonesian Journal of Computer Science, Available: https://www.researchgate.net/publication/380577026_Embracing_Distributed_Systems_for_Efficient_Cloud_Resource_Management_A_Review_of_Techniques_and_Methodologies

[6] Dhulavvagol, P.M. et al (2020) , "Performance Analysis of Distributed Processing System using Shard Selection Techniques on Elasticsearch," Procedia Computer Science, 167, Available: https://www.sciencedirect.com/science/article/pii/S1877050920308395
[7] Kleppmann M. (2015) , "A Critique of the CAP Theorem," Research Gate, Available: https://www.researchgate.net/publication/281895403_A_Critique_of_the_CAP_Theorem
[8] To Q. et al, (2018) "A Survey of State Management in Big Data Processing Systems,", The VLDB Journal, Available: https://www.researchgate.net/publication/313394436_A_Survey_of_State_Management_in_Big_Data_Processing_Systems
[9] Pamarthi S. (2024) , "Optimizing PyFlink for high-throughput machine learning: Streaming feature engineering in banking,", World Journal of Advanced Engineering Technology and Sciences, Available: https://www.researchgate.net/publication/390410591_Optimizing_PyFlink_for_high-throughput_machine_learning_Streaming_feature_engineering_in_banking