SvedbergOpen
DISSEMINATION OF KNOWLEDGE

International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: https://www.svedbergopen.com/

Research Paper Open Access

Deterministic vs. Probabilistic Models in High-Volume Stream Processing: Trade-offs in Financial and Advertising Systems

Ravindra Motiram Gurnani

Independent Researcher, USA. ORCID ID: 0009-0005-5122-5554

Abstract
High-volume data streams in financial trading and programmatic advertising routinely deliver millions of events per second, imposing constraints on memory, latency, and throughput that render exact, deterministic computation impractical in many operational contexts. This paper examines the fundamental trade-offs between deterministic and probabilistic approaches to stream processing, with emphasis on three probabilistic data structures that have achieved broad industrial adoption: HyperLogLog (HLL) for cardinality estimation, Bloom filters for set-membership testing, and Count-Min Sketch (CMS) for frequency estimation. For each structure we discuss the accuracy trade-offs and memory efficiency relative to deterministic counterparts, and situate their application within realistic financial and advertising workloads. We introduce a comparative framework that maps workload requirements—cardinality of the key space, acceptable error tolerance, regulatory constraints, and latency budgets—to the appropriate computational paradigm. Our analysis demonstrates that hybrid architectures, in which probabilistic structures handle hot-path approximation while deterministic subsystems maintain audit-grade records, offer the most pragmatic solution for regulated industries. The paper further considers integration patterns with widely deployed stream-processing runtimes, including Apache Flink and Apache Kafka, and discusses operational factors such as register sizing, hash-function selection, and counter saturation. The paper closes by outlining unresolved challenges, notably the need to address concept drift in deployed models and to meet interpretability standards demanded by financial oversight bodies.

Keywords: Stream Processing, Probabilistic Data Structures, Hyperloglog, Bloom Filter, Count-Min Sketch, Financial Systems, Advertising Technology, Approximate Computing

I. Introduction

Modern financial markets and programmatic advertising ecosystems represent two of the most demanding stream-processing environments in contemporary computing. Equity trading venues process in excess of one billion messages on peak trading days [7], while real-time bidding (RTB) platforms evaluate hundreds of thousands of ad auction requests per second across geographically distributed infrastructure [6]. Both domains share a structural characteristic: events arrive in unbounded, potentially out-of-order sequences, and decisions must be produced within strict latency budgets—often measured in single-digit milliseconds or less.

The classical response to such scale requirements has been horizontal partitioning and in-memory computation, exemplified by frameworks such as Apache Storm [10], Apache Flink [12], and Spark Streaming [9]. While these runtimes address distribution and fault tolerance, the fundamental challenge of maintaining per-key state over high-cardinality key spaces remains. A naively exact approach—maintaining precise counters, sorted structures, or hash maps for every observed entity—requires memory proportional to the cardinality of the key space and the number of distinct metrics, denoted $O(N \cdot S)$ where N is the number of distinct keys and S is the number of tracked statistics per key. At the scale of internet advertising, where hundreds of millions of user identifiers may be active within a single day, this cost becomes prohibitive.

Probabilistic data structures offer a compelling alternative: they trade a controlled, mathematically bounded error for dramatic reductions in memory footprint and computational cost. The theoretical foundations of this approach trace to the work of Flajolet and Martin [4] on probabilistic counting, and have since been refined into a family of practically deployable structures [1][2][3]. However, probabilistic approaches are not universally appropriate. Financial regulators in jurisdictions including the United States, European Union, and United Kingdom impose requirements for exact transaction records, necessitating deterministic accounting at the point of persistence even when approximation is used for real-time monitoring.

This paper makes the following contributions: (1) a systematic characterization of deterministic and probabilistic stream-processing paradigms with formal complexity bounds; (2) a concise mathematical treatment of HyperLogLog, Bloom filters, and Count-Min Sketch with their error guarantees; (3) a comparative framework mapping workload characteristics to the appropriate paradigm; and (4) a discussion of hybrid system architectures that satisfy both operational performance and regulatory compliance requirements.

II. Background

A. Data Stream Processing Models

The data stream model, formalized by Babcock et al. [20], characterizes computation over sequences of data items that arrive continuously, cannot be stored in full, and must be processed in one or a small number of passes. Formally, a stream σ is a sequence of tuples (σ_1, σ_2, …, σt), where each tuple carries a key, a value, and a timestamp. The stream is potentially unbounded, and the computation must produce answers using working memory $w \ll |\sigma|$.

Stream-processing systems impose window semantics to scope aggregations temporally. The Dataflow model [7] provides a unified treatment of windowing strategies—tumbling, sliding, and session windows—and their interaction with out-of-order event delivery and watermarking. Apache Kafka [13] has become the dominant durable transport layer for such systems, providing partitioned, replicated, and replayable log semantics that decouple producers from downstream processing stages.

B. Deterministic versus Probabilistic Paradigms

Deterministic stream-processing algorithms guarantee that every output is an exact function of the input. They are characterized by correctness in the strict sense: given the same input stream, the algorithm always produces the same output. The cost of this guarantee is typically O(N) or O(N log N) memory with respect to the cardinality of the domain, and sensitivity to adversarial or skewed input distributions [14][15].

Probabilistic algorithms relax the exactness requirement, admitting bounded errors with quantifiable probability. Muthukrishnan [5] provides a comprehensive survey of streaming algorithms in this class. The key insight is that many practically important queries—cardinality of a user population, membership of an identifier in a known fraud list, frequency of a transaction pattern—do not require exact answers; they require answers that are accurate to within a tolerance that is small relative to the decision threshold. Probabilistic structures exploit hash-based randomization to achieve this accuracy while consuming memory that is orders of magnitude smaller than deterministic equivalents.

III. Deterministic Approaches to Stream Processing

A. Exact Counting and Hash-Map Aggregation

The simplest deterministic approach to frequency tracking maintains a hash map from keys to counters. For a stream over domain D with cardinality $|D| = N$ and S tracked statistics per key, this structure requires $O(N \cdot S)$ space. In financial contexts, N may represent the number of active securities, accounts, or order identifiers. For a mid-sized broker-dealer tracking position, exposure, and volume across a large number of instruments with multiple statistics each, the memory requirement may be manageable. For an advertising platform tracking conversion events across hundreds of millions of cookie or device identifiers, the structure becomes infeasible within a single processing node.

Exact counting further requires that all update operations be serialized or partitioned to avoid race conditions in concurrent settings. Partitioning by key hash distributes the memory cost but introduces cross-partition communication overhead for operations that require global aggregates, such as computing the total number of distinct users who have been served a given advertisement campaign across all partitions.

B. Sorted Merge and Windowed Exact Aggregation

When bounded-window exactness is a hard requirement, pipelines typically employ a window-scoped key sort followed by a single-pass aggregation sweep. Spark-based batch pipelines [9] use this pattern and are well-suited to workloads whose acceptable end-to-end latency is measured in seconds rather than milliseconds. For stateful stream processors such as Apache Flink, RocksDB [8] — a log-structured merge-tree embedded store — provides exact per-key state persistence with bounded memory by progressively compacting older data into tiered storage layers.

The memory implication of deterministic windowed aggregation is that the working set must fit within available memory for the window duration or be externalized to disk with associated I/O latency penalties. For high-cardinality streams with short windows, the ratio of useful aggregate size to raw working-set size is poor, motivating approximate alternatives.

IV. Probabilistic Data Structures for Stream Processing

A. HyperLogLog: Cardinality Estimation

Cardinality estimation—computing the number of distinct elements in a stream—is a fundamental primitive in both financial and advertising analytics. Financial applications include counting distinct counterparties in a settlement window or distinct order identifiers routed through a venue. Advertising applications include estimating the reach of a campaign (the number of distinct users exposed to at least one impression) and computing frequency caps across distributed ad servers.

The HyperLogLog algorithm, introduced by Flajolet et al. [1], achieves near-optimal cardinality estimation using a fixed-size register array. Each incoming element is hashed to a uniform random bit string; the position of the leftmost 1-bit in the hash encodes information about the maximum run of leading zeros observed, which probabilistically encodes the cardinality of the stream. With m registers, the algorithm achieves a standard error of:

$$\varepsilon = 1.04 / \sqrt{m}$$

where m is the number of registers [1]. The memory requirement is O(log log N) per register, where N is the estimated cardinality, and O(m · log log N) for the full structure. In practice, a register array of m = 4,096 entries suffices for cardinality estimates across very large populations, with the standard error determined by the formula above. This is a reduction of many orders of magnitude relative to the O(N) memory required by a deterministic hash set.

The algorithm supports mergeable sketches: HLL structures computed independently on disjoint stream partitions can be unioned by taking the per-register maximum, enabling distributed cardinality estimation without cross-node coordination during the hot path. This property is particularly valuable in advertising systems where impression data is processed across geographically distributed data centers.

B. Bloom Filters: Approximate Set Membership

A Bloom filter [2] encodes a set S as a compact bit array that enables fast membership queries at the cost of a tunable false-positive probability and zero false-negative probability. The structure pairs a fixed-length bit array of m positions, initially all zero, with k independent hash functions that each map universe elements to array positions. Adding an element x marks the k positions $h_1(x)$ through hk(x) as one. To query whether y belongs to S, all k positions are checked: if every position holds a one the filter reports a probable member; if any position is zero the answer is a certain non-member.

The false-positive probability for a Bloom filter that has had n elements inserted is:

$$p \approx (1 - e^{(-kn/m)})^k$$

The optimal number of hash functions that minimizes this probability for given m and n is:

$$k^* = (m/n) \cdot \ln 2$$

as derived in [2]. The memory requirement is O(m) bits, where m is selected based on the target false-positive rate and expected set size n, independent of the domain size. This fixed-space property makes Bloom filters well-suited to financial fraud detection pipelines, where a set of known fraudulent account identifiers—potentially numbering in the millions—must be tested against every incoming transaction. A deterministic hash set of the same population would require O(n) space proportional to the number of fraudulent identifiers plus the overhead of the hash table, while a Bloom filter with a target false-positive rate approaching zero requires only a modest fixed allocation in bits per element.

In advertising, Bloom filters are used for frequency capping: rather than maintaining exact impression counts per user per campaign, a filter can rapidly determine whether a user has received no impressions (definite non-member) versus possibly having received impressions, subject to the configurable false-positive bound. The asymmetric error—no false negatives, bounded false positives—aligns naturally with the asymmetric cost structure of advertising delivery, where under-delivery (missing a real user) is typically more costly than a rare excess delivery to a misidentified user.

C. Count-Min Sketch: Frequency Estimation

Cormode and Muthukrishnan [3] introduced the Count-Min Sketch (CMS) as a compact frequency oracle for data streams with formally bounded additive error. Its internal layout is a rectangular counter grid with d independent rows, each holding w cells, giving d × w counters in total. Every row is paired with its own hash function that maps incoming stream elements to cell indices within [1, w]. When an element x arrives, each of the d hash functions is applied and the corresponding counter incremented; the frequency estimate returned for x is the smallest of those d counter values, which by the minimum operator suppresses overcount from hash collisions.

The algorithm guarantees that the estimate $\hat{f}(x)$ satisfies:
$f(x) \leq \hat{f}(x) \leq f(x) + \varepsilon \cdot ||f||_1$
with probability at least $1 - \delta$, where $||f||_1$ denotes the L1 norm of the frequency vector (the total stream length), ε is the additive error parameter, and δ is the failure probability. The sketch dimensions are set as $w = \lceil e/\varepsilon \rceil$ and $d = \lceil \ln(1/\delta) \rceil$, where e denotes the base of the natural logarithm [3]. The memory requirement is $O(d \cdot w) = O((1/\varepsilon) \cdot \log(1/\delta))$, which is polylogarithmic in the stream length and independent of the domain cardinality.
In financial stream processing, CMS is applicable to detecting heavy hitters—securities, accounts, or order types that account for disproportionate fractions of trading volume—within a rolling window. Identifying heavy hitters in near-real time informs risk management decisions and circuit-breaker logic. In advertising, CMS supports budget pacing: frequency estimates for ad campaign impressions allow a pacing controller to modulate delivery rates without maintaining exact counters for each of the potentially billions of active campaign–user combinations.
The structure is naturally amenable to distributed aggregation. CMS sketches computed on independent stream partitions can be merged by element-wise addition, analogously to HLL union. This composability, established in [3] and further analyzed in [14], enables hierarchical aggregation architectures where edge nodes maintain local sketches and a central aggregator merges them periodically.

V. Comparative Analysis

A. Formal Complexity Comparison

Table I summarizes the space complexity, query time complexity, error characteristics, and mergeability of the approaches discussed in the preceding sections. N denotes the cardinality of the key domain, n the number of inserted elements, m the structure size parameter, k the number of hash functions, ε the additive error parameter, and δ the failure probability.

TABLE I — Complexity and Correctness Comparison

Model	Memory Complexity	Compute per Event	Exactness Guarantee	Error Rate	Domain Suitability
Exact Set (HashMap)	O(N·S)	O(1) amortized	Exact	0%	Financial, audit-required
RocksDB State (Flink)	O(N·S) disk-spill	O(log N) read / O(1) write	Exact (with 2PC)	0%	Financial transaction, regulatory reporting
Bloom Filter	O(m) bits	O(k) — k hash evals	Probabilistic, one-sided	p≈(1−e^(−kn/m))^k	Existence checking, dedup pre-filter
HyperLogLog	O(2^b) registers	O(1) — single hash + register	Probabilistic	1.04/√m	Cardinality estimation, unique reach
Count-Min Sketch	O(d·w)	O(d) — d hash evals	Probabilistic, bounded	ε w.p. 1−δ	Frequency estimation, heavy-hitter detection

B. Trade-off Framework

The selection of a deterministic or probabilistic approach is governed by four workload dimensions: (1) key-space cardinality, (2) tolerable error bound, (3) regulatory or audit constraints, and (4) latency budget.
When the key space is small (N below approximately one million), deterministic hash maps are computationally feasible and preferred for correctness. As N scales into the hundreds of millions, probabilistic structures

become necessary to avoid exceeding node memory limits. The transition point depends on available hardware and the number of statistics tracked per key.

Tolerable error bound is application-specific. A campaign reach estimate used to adjust bidding strategy tolerates errors of one to two percent; a settlement ledger entry does not. Regulatory constraints impose hard requirements for deterministic exactness in specific contexts regardless of operational convenience, as discussed in Section VII.

Latency budgets favor probabilistic structures in two ways. First, the smaller memory footprint of probabilistic structures improves cache locality, reducing average memory access latency. Second, the absence of coordination requirements for mergeable sketches eliminates the synchronization overhead that deterministic approaches incur when aggregating across partitions [7].

TABLE II — Production Performance Profiles at 10^6 Events/sec

Model	Est. Memory at 10^8 Unique Keys	Est. Latency per Lookup	Throughput (ev/s)	Correctness Loss	Infra Cost Tier
Exact Set	~32 GB (320-byte avg record)	O(1) sub-microsecond	10^6–10^7	None	High
RocksDB/Flink	Tens of GB RAM + SSD spill	O(log N) microsecond	10^6–3×10^6	None (with 2PC)	High
Bloom Filter	~958 MB at 1% FP for 10^8 elements	O(k) nanosecond	>10^7	FP at rate p	Low-Medium
HyperLogLog	~12 KB (b=14 registers)	O(1) nanosecond	>10^7	~0.81% std error	Low
Count-Min	O(d·w) — typically MBs	O(d) nanosecond	>10^7	Bounded overcount	Low

TABLE III — Domain-Driven Decision Matrix

Domain	Correctness Req. Type	Reg./Commercial Basis	Error Tolerance	Recommended Model	Consequence of Wrong Choice
Financial ledgering	Legal-regulatory	Banking/payments regulation	Zero	Deterministic (RocksDB/2PC)	Regulatory non-compliance; audit failure
Financial fraud detect.	Legal-regulatory	AML/BSA compliance	Near-zero	Deterministic (exactly-once)	Undetected fraud; AML failures
AdTech reach measurement	Commercial	IAB standards; advertiser SLAs	<2% std error	Probabilistic (HyperLogLog)	Deterministic is cost-prohibitive at scale
AdTech freq. capping	Commercial	Campaign contracts; platform SLAs	Small overcount	Probabilistic (Count-Min Sketch)	Deterministic unaffordable at ad cardinalities
Mixed FinTech/AdTech	Per-feature	Feature-by-feature classification	Varies	Hybrid (domain-partitioned)	Miscategorization contaminates

Domain	Correctness Req. Type	Reg./Commercial Basis	Error Tolerance	Recommended Model	Consequence of Wrong Choice
					correctness posture

VI. System Design Implications

A. Integration with Apache Kafka and Apache Flink

A representative high-volume stream-processing architecture for financial or advertising workloads consists of a durable event log (Apache Kafka [13]) feeding a stateful stream processor (Apache Flink [12]) that maintains operator state in embedded storage such as RocksDB [8]. Probabilistic data structures can be embedded within Flink operator state as serializable objects, with state backends persisting snapshots for fault tolerance via Flink's distributed checkpointing mechanism.

HyperLogLog registers are naturally serializable as byte arrays and can be stored as Flink ValueState keyed by a window identifier. Merging across parallel operator instances can be performed in a reduce function that takes the per-register maximum, requiring O(m) communication per window boundary rather than the O(N) communication required to aggregate exact hash maps. This reduction in cross-node data transfer is significant in wide-area deployments where inter-datacenter bandwidth is a constraint.

Bloom filters require careful lifecycle management in streaming contexts because they are append-only structures: elements can be inserted but not removed. Sliding-window semantics therefore require either a rotation strategy—maintaining two filters and periodically discarding the older one—or a counting Bloom filter variant that supports deletions at the cost of additional memory. The rotation strategy introduces a brief period during which false-positive rates may be elevated as the new filter is sparsely populated.

Count-Min Sketch state in Flink can be maintained as a two-dimensional counter array within a ListState or MapState abstraction. Merging across partitions by element-wise addition is straightforward and supported by Flink's process function API. However, the counter width must be selected to prevent saturation: in high-volume streams, individual counters may overflow if the sketch dimensions are set conservatively. Using 32-bit or 64-bit integer counters rather than 8-bit counters increases memory consumption but avoids saturation artifacts that would violate the error bound guarantees of [3].

B. Latency Budgets and Operational Considerations

In algorithmic trading contexts, end-to-end processing latency from market-data ingestion to order generation is typically bounded at one to ten milliseconds. Probabilistic structures contribute to meeting this budget by eliminating the need for remote state lookups: an in-memory HLL register array or CMS counter grid can be updated in nanoseconds. Deterministic hash maps backed by RocksDB incur microsecond-level read and write latencies that, while acceptable in many contexts, become a bottleneck at the highest throughput levels.

Register sizing for HyperLogLog involves a direct trade-off between memory and accuracy. An operator targeting low standard error requires a larger register array, which must fit within the L3 cache of the processing node to achieve optimal access latency. For a given cache budget, the operator should select the largest register array that fits, maximizing accuracy while remaining within cache. The S4 distributed computing platform [18] and its successors demonstrated that in-process state management, which avoids serialization overhead, is essential for achieving sub-millisecond update latencies in stream-processing operators.

VII. Discussion

A. Hybrid Architectures

The dichotomy between deterministic and probabilistic processing is, in practice, not a binary choice but a design dimension along which different components of a system may be positioned independently. A financially sound architecture for a trading platform may use CMS-based heavy-hitter detection on the hot path to identify instruments warranting elevated risk scrutiny, while simultaneously persisting exact event records to a deterministic store for regulatory reporting. The probabilistic component operates at stream speed; the deterministic component operates at storage speed on the same event log, lagging by at most a few seconds.

This dual-path pattern is consistent with the lambda architecture concept and its stream-native successors [7], but it requires that both paths consume the same source-of-truth event log. Apache Kafka's consumer group semantics [13], which allow multiple independent consumers to replay the same log from independent offsets,

are well-suited to this requirement. The probabilistic consumer processes events in real time for monitoring and alerting; the deterministic consumer processes the same events with higher latency for audit-grade aggregation.

Hybrid architectures also appear at the data structure level. The DEBS 2014 grand challenge [19] benchmarked approaches to real-time analytics over high-frequency financial data and highlighted the practical value of structures that provide approximate answers during peak load with the ability to fall back to exact computation during low-load periods. Dynamic switching between modes based on current system load is an active area of systems research.

TABLE IV — Hybrid Architecture Design Patterns

Pattern Name	When to Apply	Deterministic Component	Probabilistic Component	Key Trade-off	Example System Context
Hot-Cold Split	Time-bounded exactness requirements	Exact state for recent window	Probabilistic for historical windows	Memory grows for hot window only	Financial audit + long-tail analytics
Per-Feature Correctness	Mixed workloads with different feature req.	Deterministic for regulated features	Probabilistic for non-regulated features	Two parallel state backends	FinTech payment + advertising platform
Tiered Accuracy	High-value entities need exactness; long tail OK	Exact tracking for top-N entities	Count-Min/HLL for remaining population	Threshold determination + tier-migration	Fraud systems: top accounts exact-monitored
Domain-Partitioned	Org boundary between regulated/commercial data	Separate Flink job, exactly-once	Separate probabilistic pipeline	Operational overhead of two pipelines	Large orgs: compliance + marketing analytics

B. When Determinism Is Non-Negotiable

Several categories of financial computation require deterministic exactness by regulatory mandate. Position reporting under MiFID II in the European Union and Rule 17a-3 in the United States requires exact records of all transactions and resulting positions. Anti-money-laundering systems that generate Suspicious Activity Reports must base their determinations on exact transaction histories, not approximations. These requirements constrain the use of probabilistic structures to the monitoring and alerting layer and preclude their use in the authoritative record layer.

The interpretability dimension discussed by Doshi-Velez and Kim [16] in the context of machine learning applies analogously to probabilistic data structures in regulated settings: a regulator asking why a system flagged a particular account expects a deterministic, auditable answer, not a probabilistic sketch state. This constraint reinforces the hybrid architecture pattern: probabilistic structures generate signals that trigger deterministic downstream investigation, preserving the performance benefits of approximation while meeting auditability requirements.

Concept drift—the phenomenon whereby the statistical properties of the stream change over time [21]—poses a specific challenge for probabilistic structures. A Bloom filter populated with known fraud identifiers becomes stale as the fraud population evolves; an HLL sketch initialized at the start of a trading session accumulates cardinality monotonically and cannot adapt to decreasing cardinality. Adaptive strategies such as periodic sketch reset, sliding-window filters, and sketch aging are necessary in production deployments but introduce additional engineering complexity.

C. When Approximation Is Preferred

Outside the determinism-mandatory zone, probabilistic approaches offer substantial advantages. Campaign reach measurement in advertising, where the goal is to estimate the number of unique users exposed to a creative, is a canonical use case for HyperLogLog. The UCI Machine Learning Repository [17] provides advertising datasets that illustrate the scale of user identifier spaces in this domain. Exact reach computation would require maintaining a set of all observed user identifiers within a time window—an O(N) memory requirement that is infeasible at internet scale.

Click fraud detection, which requires testing incoming click events against a list of known bot or fraudulent identifiers, is a natural fit for Bloom filters. The asymmetric error—no false negatives, bounded false positives—means that legitimate clicks from users not in the fraud list are never incorrectly blocked, while fraudulent clicks from listed identifiers are always caught. Rare false positives on legitimate users impose a minor revenue cost, whereas false negatives (missed fraud) are a direct financial loss, making the asymmetric error semantics operationally favorable.

Skewed data streams, analyzed by Cormode and Muthukrishnan [14] and Estan and Varghese [15] in the context of network traffic measurement, are also common in financial contexts: a small fraction of instruments or accounts typically accounts for a large fraction of trading volume. Count-Min Sketch's minimum estimator is particularly well-suited to such Zipf-distributed streams, as the dominant items are least likely to experience hash collisions that would inflate their estimates. Graph-join processing challenges [22] are also relevant when stream events must be enriched with graph-structured reference data during processing, a pattern common in entity-resolution pipelines for AML compliance.

VIII. Conclusion

This paper has examined the trade-offs between deterministic and probabilistic stream-processing approaches in the context of high-volume financial and advertising workloads. The central finding is that no single paradigm dominates across all workload dimensions: the appropriate choice depends on key-space cardinality, tolerable error, regulatory constraints, and latency budget.

The three probabilistic data structures surveyed—HyperLogLog, Bloom filters, and Count-Min Sketch—each address a distinct query primitive (cardinality estimation, set membership, frequency estimation) with mathematically rigorous error guarantees and memory requirements that are orders of magnitude smaller than their deterministic counterparts. Their mergeability properties make them particularly well-suited to the distributed, partitioned architectures that characterize modern stream-processing deployments on platforms such as Apache Flink and Apache Kafka.

The comparative framework introduced in Section V maps workload characteristics to structure selection and is intended as a practical reference for system designers. The hybrid architecture pattern discussed in Section VII, in which probabilistic hot-path processing coexists with deterministic audit-grade persistence on a shared event log, offers a pragmatic resolution to the tension between operational performance and regulatory compliance.

This analysis carries several limitations worth noting. The accuracy guarantees derived for each probabilistic structure presuppose hash functions that behave as truly uniform random mappings; in adversarial workloads where an attacker can engineer inputs that concentrate on hash collisions, observed error can exceed the stated theoretical ceiling. The memory figures reported likewise reflect clean mathematical models rather than production deployments, which carry additional overhead from object metadata, serialization framing, and concurrent-access locking. Three directions merit attention in future research: methods for dynamically resizing sketches in response to concept drift [21]; principled approaches for embedding probabilistic summaries into graph-structured stream-enrichment pipelines [22]; and the design of interpretability wrappers that make approximate query outputs legible to financial regulators without sacrificing the throughput gains that motivate the structures in the first place [16].

References

[1] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm," in Proceedings of the 13th Conference on Analysis of Algorithms (AofA '07), 2007, pp. 127–146. https://doi.org/10.46298/dmtcs.3545

[2] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," Communications of the ACM, vol. 13, no. 7, pp. 422–426, 1970. https://doi.org/10.1145/362686.362692

[3] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," Journal of Algorithms, vol. 55, no. 1, pp. 58–75, 2005. https://doi.org/10.1016/j.jalgor.2003.12.001

[4] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," Journal of Computer and System Sciences, vol. 31, no. 2, pp. 182–209, 1985. https://doi.org/10.1016/0022-0000(85)90041-8

[5] S. Muthukrishnan, "Data streams: Algorithms and applications," Foundations and Trends in Theoretical Computer Science, vol. 1, no. 2, pp. 117–236, 2005. https://doi.org/10.1561/0400000002

[6] D. Agarwal, B.-C. Chen, P. Elango, N. Motgi, S.-T. Park, R. Ramakrishnan, S. Roy, and J. Zachariah, "Online models for content optimization," in Advances in Neural Information Processing Systems, vol. 21, 2008.

[7] T. Akidau et al., "The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," Proceedings of the VLDB Endowment, vol. 8, no. 12, pp. 1792–1803, 2015. https://doi.org/10.14778/2824032.2824076

[8] S. Dong et al., "RocksDB: Evolution of development priorities in a key-value store serving large-scale applications," ACM Transactions on Storage, vol. 17, no. 4, pp. 1–32, 2021. https://doi.org/10.1145/3483840

[9] M. Zaharia et al., "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in Proc. 9th USENIX Symposium on Networked Systems Design and Implementation, 2012.

[10] A. Toshniwal et al., "Storm @Twitter," in Proc. 2014 ACM SIGMOD International Conference on Management of Data, 2014, pp. 147–156. https://doi.org/10.1145/2588555.2595641

[11] S. Kulkarni et al., "Twitter Heron: Stream processing at scale," in Proc. 2015 ACM SIGMOD International Conference on Management of Data, 2015, pp. 239–250. https://doi.org/10.1145/2723372.2742788

[12] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache Flink: Stream and batch processing in a single engine," IEEE Data Engineering Bulletin, vol. 38, no. 4, pp. 28–38, 2015.

[13] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in Proc. NetDB Workshop, 2011.

[14] G. Cormode and S. Muthukrishnan, "Summarizing and mining skewed data streams," in Proc. 2005 SIAM International Conference on Data Mining, 2005, pp. 44–55. https://doi.org/10.1137/1.9781611972757.5

[15] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," ACM Transactions on Computer Systems, vol. 21, no. 3, pp. 270–313, 2003. https://doi.org/10.1145/859716.859719

[16] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," arXiv preprint arXiv:1702.08608, 2017.

[17] M. Lichman, "UCI Machine Learning Repository," University of California, Irvine, School of Information and Computer Sciences, 2013. https://archive.ics.uci.edu/

[18] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in Proc. 2010 IEEE International Conference on Data Mining Workshops, 2010, pp. 170–177. https://doi.org/10.1109/ICDMW.2010.172

[19] Z. Jerzak and H. Ziekow, "The DEBS 2014 grand challenge," in Proc. 8th ACM International Conference on Distributed Event-Based Systems, 2014, pp. 266–269. https://doi.org/10.1145/2611286.2611333

[20] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in Proc. 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, 2002, pp. 1–16. https://doi.org/10.1145/543613.543615

[21] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," ACM Computing Surveys, vol. 46, no. 4, pp. 1–37, 2014. https://doi.org/10.1145/2523813

[22] D. Nguyen, W. Kim, and H. Shim, "Join processing for graph patterns: An old dog with new tricks," in Proc. 2015 ACM SIGMOD International Conference on Management of Data, 2015, pp. 547–562. https://doi.org/10.1145/2723372.2723732