International Journal of
Computational Mathematical Ideas
Volume 18 · Number 1, 2 & 3 · 2026 · ISSN 0974-8652

Available online at www.ijcmi.com

Data Engineering Architectures and Token-Efficient Tooling for Large-Scale Event Handling in Modern Agentic Artificial Intelligence Systems

Saisuman Singamsetty *

Data Management Specialist,

American Unit Inc, San Antonio, TX, USA

*Corresponding author: sudheersingamsetty99@gmail.com

Article Info

Article history:
Received: 05-01-2026
Revised: 28-06-2026
Accepted: 18-07-2026
Available online: 2026

Keywords:
Data engineering; Event stream processing; Agentic AI; Token optimization; Streaming sketches; Windowed aggregation; Modern data tooling.

Abstract

Modern agentic artificial intelligence systems increasingly reason over high volume enterprise event streams that arrive from clickstreams, sensors, transactions, and application telemetry. Two forces meet in these systems. Data engineering must ingest and reduce millions of events per interval, and the language model that drives the agent must observe those events within a strict token budget. This paper connects the two concerns with a single mathematical and algorithmic treatment of large scale event handling for token efficient agents. We model the reduction pipeline as a sequence of mergeable window operators and streaming sketches, and we formulate the agent facing cost as a token budget problem in which data engineering choices set the size of every observation. We implement the full pipeline from first principles and report reproducible measurements on a stream of one million synthetic events across four domains. A single reduction pass processes about 2.6 million events per second, tumbling window aggregation reduces row counts by factors between 1.80 and 2.11, HyperLogLog cardinality estimates track the theoretical one point zero four over root m error from 11.7 percent at 64 registers down to 0.81 percent at 4096 registers, a Count Min sketch lowers heavy hitter error from 34.8 percent to 0.32 percent as its width grows, and reservoir sampling holds the sampled domain mix within 0.079 percent of the population. Feeding an agent windowed summaries rather than raw events cuts the observation token count by 39.1 percent, and a transparent episode cost model shows that this reduction lowers per episode token consumption by 38.7 percent for an eight round event driven agent. The results give practitioners a rigorous basis for pairing data engineering with token budgets in agentic deployments.

1. Introduction

Agentic artificial intelligence systems have moved from answering isolated questions to driving continuous enterprise workflows. A modern agent plans a task, calls tools, retrieves evidence, and acts on the world across several reasoning rounds, and in an operational setting the evidence it must weigh is rarely a static document. It is a live stream of events: user clicks, sensor readings, payment records, and application logs that arrive by the millions every few minutes. The agent is only as good as its view of that stream, and that view is expensive twice over. It is expensive on the data plane, where a pipeline must ingest, clean, deduplicate, and aggregate a torrent of records in near real time, and it is expensive on the model plane, where every event that reaches the language model consumes tokens, which are the unit of both latency and cost. These two expenses are usually studied by two different communities. Data engineering optimizes throughput and storage, while agent research optimizes prompts and reasoning. This paper argues that for large scale event handling the two are one problem, because the data engineering decision of how aggressively to reduce a stream is exactly the decision that sets the token size of every observation the agent sees.

The value of agentic pipelines over enterprise data is already visible across functions. Autonomous portfolio agents that combine reasoning with market interaction improve stock selection and execution in algorithmic trading [20]. Reinforcement learning controllers optimize agile portfolio management on the Nifty 50 index [21] and maximize

profit in perishable food supply chains [22]. Cross layered edge and cloud deep learning supports digital twin analytics in smart manufacturing [23]. Each of these settings runs on a stream, and in each the economic viability of the agent depends on how frugally it converts raw events into tokens, which is the subject of this paper.

We take a deliberately unified view. We treat the reduction pipeline as a composition of mergeable window operators and streaming sketches, we prove the structural properties that let those operators run in parallel and in bounded memory, and we measure the resulting token savings when the reduced stream is handed to an agent. Concretely this paper makes four contributions.

1. A formulation that casts large scale event handling for agents as a token budget problem, in which mergeable window aggregation, cardinality and frequency sketches, and uniform sampling are the operators that set the size of each agent observation (Sections 3 and 4).
2. A from scratch implementation and a reproducible measurement of the data engineering layer on a stream of one million synthetic events across four enterprise domains, reporting throughput, reduction ratio, and duplicate rate (Section 6).
3. An accuracy study of the streaming sketches against exact ground truth, showing that HyperLogLog tracks its theoretical error and that a Count Min sketch and a reservoir sample stay within tight bounds (Section 6).
4. A transparent parametric cost model, calibrated on measured tokenizer output, that quantifies how much windowed summarization lowers the token cost of an event driven agent across reasoning rounds (Section 6).

2. Background and Related Work

2.1. Large scale event processing and modern tooling

The batch model of computation over massive data was popularized by MapReduce, which expressed computation as a map step followed by a reduce step over key grouped data [8]. In memory cluster computing then reduced the cost of iterative and interactive workloads through resilient distributed datasets in Apache Spark [9]. For unbounded data the community moved from batch to stream processing, where a continuous query runs over records as they arrive. Apache Flink provides stateful stream processing with exactly once semantics [10], and a durable partitioned log such as Apache Kafka serves as the transport layer that decouples producers from consumers [11]. The semantics of windowing, triggers, and watermarks that reconcile event time with processing time were unified in the dataflow model [12]. The foundational abstractions and open problems of processing data that is seen only once were set out early in the data stream literature [13]. Table formats that give streaming and batch engines transactional views over object storage, such as Delta Lake, complete the modern stack [14]. Our pipeline is a compact, self contained realization of these ideas built so that every number in the paper is reproducible offline.

2.2. Streaming sketches and approximate aggregation

When a stream is too large to store, exact answers give way to compact probabilistic summaries. HyperLogLog estimates the number of distinct elements in a single pass using a small register array, with a relative standard error of about $1.04/\sqrt{m}$ for m registers [15]. The Count Min sketch answers point and frequency queries in sublinear space and improves the space to accuracy trade off of earlier summaries [16]. Uniform sampling without knowing the stream length in advance is solved by reservoir sampling, which keeps each prefix element with the correct probability [17]. A broad account of why such sketches matter for practical systems is given in the data sketching literature [18]. These structures are the mathematical core of the reduction layer we analyse, because they let a pipeline answer the aggregate questions an agent actually asks without materializing the full stream.

2.3. Agentic AI and token bounded observation

The transformer architecture underpins the models that read and generate token sequences [1], and large models exhibit strong few shot behaviour that makes them attractive agent controllers [2]. Agentic methods interleave reasoning and acting, as in the ReAct framework [3], and let the model invoke external tools through learned interfaces [4]. Retrieval augmented generation supplies external evidence at inference time and thereby shifts cost onto the size of the retrieved context [5]. A recent survey catalogues the rapid growth of language model based autonomous agents [19]. In all of these the observation an agent reads is paid for token by token, so the representation of event data is a first order design choice. To measure that representation honestly we tokenize with byte pair encoding, which originated as a compression scheme [6] and became the standard subword method for neural models [7]. The present work is complementary to prior agentic systems built by the author and colleagues, including privacy preserving big data publishing on Hive [25], a sensitivity driven publishing framework that combines cache and retrieval augmented

generation [24], and classifiers for imbalanced data at scale on Hadoop [26]. It differs in placing the token cost of the agent, rather than storage or model accuracy alone, at the centre of the pipeline design.

3. Notation and Problem Formulation

We model a stream as an ordered sequence of timestamped events and study operators that reduce it before it reaches an agent.

Definition 1 (Event stream)**.** *An event stream is a sequence $S = (e_1, e_2, \dots)$ where each event $e_i = (t_i, \kappa_i, x_i)$ carries a timestamp t_i, an entity key $\kappa_i \in \mathcal{K}$, and a payload x_i. Events arrive in approximately non decreasing timestamp order and the length of S is not known in advance.*

Definition 2 (Tumbling window)**.** *For a fixed width $\Delta > 0$ the window index of event e_i is $w_i = \lfloor t_i/\Delta \rfloor$. A tumbling window aggregation groups events by the pair (κ_i, w_i) and emits one summary row per non empty group.*

Let $n = |S|$ be the number of events processed in an interval and let $A(S)$ be the multiset of aggregate rows produced by an operator. The reduction ratio is

$$\rho(S) \;=\; \frac{n}{|A(S)|}, \tag{1}$$

so that larger ρ means a more compact summary. When the agent reads a representation r of the stream, its token cost is $\tau(r)$, the number of subword tokens produced by the tokenizer. For two representations r_{raw} and r_{agg} of the same interval we define the token reduction

$$\eta \;=\; 1 - \frac{\tau(r_{\text{agg}})}{\tau(r_{\text{raw}})}. \tag{2}$$

An agent runs for R rounds. In round r it reads a system preamble of T_{sys} tokens once, an observation, and produces reasoning and action tokens T_{think} and T_{act}. Observations accumulate in the context because earlier evidence is replayed, and their per round size grows geometrically with a factor $g \geq 1$ as the task deepens. If the base observation is T_{obs} tokens and the data engineering layer scales it by a factor $\kappa \in (0, 1]$, the cumulative token cost of an episode is

$$C(\kappa) \;=\; T_{\text{sys}} + \sum_{r=1}^{R} \Big(T_{\text{think}} + T_{\text{act}} + \kappa\, T_{\text{obs}} \sum_{j=1}^{r} g^{\,j-1} \Big). \tag{3}$$

The inner sum is geometric, so $\sum_{j=1}^{r} g^{\,j-1} = (g^r - 1)/(g - 1)$ for $g > 1$, which makes the context term the dominant, super linear part of the cost. The optimization problem we study is to choose reduction operators that minimize $C(\kappa)$ subject to a fidelity constraint that the summary preserve the aggregate quantities the task needs, namely counts, per group statistics, distinct counts, and heavy hitters, each to a stated tolerance.

4. Mathematical Methods

4.1. Mergeable window aggregation

The window operator is attractive because it is associative, which is what lets a pipeline compute it in parallel across shards and combine partial results. Consider aggregates of the form count, sum, minimum, maximum, and last value. Each is an element of a commutative monoid with a merge operation $\oplus$ and an identity.

Proposition 1 (Parallel mergeability)**.** *Let a per group aggregate be a tuple $a = (n, \sigma, \ell, u)$ recording count, sum, minimum, and maximum, with merge*

$$a \oplus a' = \big(n + n',\; \sigma + \sigma',\; \min(\ell, \ell'),\; \max(u, u')\big).$$

Then $\oplus$ is associative and commutative with identity $(0, 0, +\infty, -\infty)$. Consequently, for any partition of the events of a group across shards, aggregating each shard and merging the partial results yields the same tuple as aggregating the group in one pass.

Proof. Each component is built from operations that are themselves associative and commutative: integer and real addition for n and σ, and the lattice operations min and max for ℓ and u. A tuple of associative and commutative operations is associative and commutative componentwise, and the stated identity leaves every component unchanged. Associativity implies that the result is independent of the order and grouping in which partial aggregates are merged, which is exactly the claim. □

Mergeability guarantees that the reduction ratio ρ is achievable at scale without a global sort, because groups are formed by hashing (κ, w) and merged locally. The same property underlies the windowing semantics of modern streaming engines [12].

4.2. Cardinality estimation

A recurring aggregate is the number of distinct entities in a window, for example distinct users or devices. Exact counting needs memory proportional to the number of distinct keys, which defeats the purpose of streaming. HyperLogLog instead hashes each key uniformly into one of $m = 2^p$ registers and records, per register, the maximum number of leading zeros seen in the remaining hash bits [15]. Writing M_j for the value in register j, the estimator is

$$\hat{D} \;=\; \alpha_m m^2 \Big(\sum_{j=1}^{m} 2^{-M_j} \Big)^{-1}, \qquad \alpha_m = \Big(m \int_0^{\infty} \big(\log_2 \tfrac{2+u}{1+u} \big)^m du \Big)^{-1}, \tag{4}$$

with a small range correction when many registers are empty. The harmonic mean form suppresses the influence of large outlier registers, and the standard error of $\hat{D}$ is approximately $1.04/\sqrt{m}$, so accuracy improves as the square root of the memory. This square root law is the lever a pipeline pulls to trade a controllable error for a fixed, tiny memory footprint, and we verify it directly in Section 6.

4.3. Heavy hitters

Agents often need the most active entities rather than a full frequency table. The Count Min sketch maintains a table of d rows and w columns, hashing each key with d independent functions and incrementing the corresponding cell in each row [16]. The estimate for a key is the minimum of its d cells, which never underestimates the true count because every true occurrence increments each of the key cells. With $w = \lceil e/\varepsilon \rceil$ and $d = \lceil \ln(1/\delta) \rceil$ the estimate $\hat{f}_x$ for key x satisfies

$$f_x \;\le\; \hat{f}_x \;\le\; f_x + \varepsilon \, \|f\|_1 \quad \text{with probability at least } 1-\delta, \tag{5}$$

where $\|f\|_1$ is the total stream length. The additive error is thus controlled by the width, and because heavy hitters have large f_x their relative error shrinks quickly as w grows, which the experiment confirms.

4.4. Uniform reservoir sampling

When a task needs a representative raw sample, for example to let the agent inspect exemplar events, the pipeline must draw a uniform sample of fixed size k from a stream of unknown length. Reservoir sampling keeps the first k events, then for the ith event with $i > k$ replaces a uniformly chosen reservoir slot with probability k/i [17].

Proposition 2 (Uniformity)**.** *After processing $n \ge k$ events, every event has probability exactly k/n of being in the reservoir.*

Proof. We argue by induction on n. For $n = k$ every event is retained with probability $1 = k/k$. Assume the claim holds after $n-1$ events, so each has probability $k/(n-1)$. The nth event enters the reservoir with probability k/n, which establishes the claim for it. For any earlier event, it remains if it was present, with probability $k/(n-1)$, and is not evicted by the nth event. The nth event causes an eviction with probability k/n, and given an eviction each of the k slots is equally likely, so a fixed present element is evicted with probability $(k/n)(1/k) = 1/n$. Hence it survives with probability $1 - 1/n = (n-1)/n$, and its total probability is $\frac{k}{n-1} \cdot \frac{n-1}{n} = \frac{k}{n}$, completing the induction. □

Uniformity means the sampled stream inherits the population's mix of event types and entities, so the agent's spot checks are unbiased while its token cost stays fixed at the size of k events.

4.5. Token cost of event driven agents

The methods above all lower $|A(S)|$ or fix the observation size, and through the tokenizer they lower τ. Substituting the measured scale factor $\kappa = \tau(r_{\text{agg}})/\tau(r_{\text{raw}})$ into the episode cost (3) propagates a one time data engineering choice through every round. Because the context term grows as g^r, a constant factor reduction in the per round observation compounds into a large absolute saving over an episode, which is the effect we quantify next.

5. Algorithms in Agentic Data Pipelines

Algorithm 1 states the token aware ingestion loop that ties the operators together. Each micro batch of events is reduced by mergeable window aggregation, summarized by the sketches for distinct counts and heavy hitters, and optionally sampled for exemplars. Only the reduced artefacts are serialized into the agent's observation, which keeps the token cost proportional to the number of active groups rather than the number of raw events.

Algorithm 1. *Token aware event ingestion for an agent*

Input: stream S, window width Δ, sketch sizes (m, w, d), sample size k
Init: aggregate map $\mathscr{A} \leftarrow \emptyset$; HyperLogLog H; Count Min C; reservoir $\mathscr{R}$

1. **for** each event $e = (t, \kappa, x)$ in the micro batch **do**
2. $\quad w_e \leftarrow \lfloor t/\Delta \rfloor$; merge e into $\mathscr{A}[(\kappa, w_e)]$ // monoid merge, Prop. 1
3. $\quad$ update H with κ; update C with κ // distinct and frequency
4. $\quad$ offer e to reservoir $\mathscr{R}$ with the k/i rule // uniform sample, Prop. 2
5. $\hat{D} \leftarrow H$.ESTIMATE(); $\mathscr{Q} \leftarrow C$.HEAVYHITTERS()
6. serialize a compact summary from $\mathscr{A}$, $\hat{D}$, $\mathscr{Q}$, and $\mathscr{R}$
7. feed the summary as the agent observation; the agent reasons and acts
8. **return** agent action and the retained sketches for the next batch

Figure 1. A token aware ingestion loop. Only the reduced artefacts of lines 5 and 6 reach the agent, so the observation token cost scales with the number of active groups rather than the number of raw events.

The loop maps cleanly onto modern tooling. A partitioned log delivers the micro batches [11], a stream processor maintains the windowed state and sketches with exactly once semantics [10], the windowing follows the dataflow model's event time and trigger semantics [12], and a transactional table format persists the reduced rows for replay and audit [14]. The agent itself follows the reason and act pattern [3] and reads only the serialized summary, so the model plane never sees the raw torrent.

6. Experimental Analysis

6.1. Setup

We generate a stream of one million events across four enterprise domains, namely clickstream, sensor telemetry, financial transactions, and application logs, with entity popularity drawn from a Zipf like distribution so that a small number of entities dominate traffic. All randomness uses a fixed seed. We implement the window aggregation, HyperLogLog, Count Min sketch, and reservoir sampling from first principles, and we tokenize event representations with a byte pair encoder trained from scratch on the event text so that token counts are genuine rather than assumed. The code emits a machine readable results file from which every figure and table in this section is drawn.

6.2. Reduction and throughput

Table 1 reports, per domain, the event count, raw serialized size, the exact number of distinct entities, the duplicate rate, the number of tumbling window rows at a sixty second width, and the resulting reduction ratio. A single reduction pass over the full stream processes about 2.58×10^6 events per second in pure interpreted code, and the windowed aggregation compresses the row count by factors between 1.80 and 2.11 depending on how bursty the domain is. Clickstream, with the highest duplicate rate at 16.11 percent, benefits most from the combination of deduplication and aggregation. Figure 2 visualizes the reduction ratio against the duplicate rate across domains.

Table 1. Per domain stream statistics and window reduction on one million events.

Domain	Events	Raw (KB)	Distinct	Dup. (%)	Reduction
Clickstream	400889	38460.0	40483	16.11	2.11×
IoT telemetry	299456	30614.6	7996	4.56	2.03×
Transaction	150333	13643.9	18789	0.02	1.80×
Application log	149322	13918.6	1200	1.63	1.85×

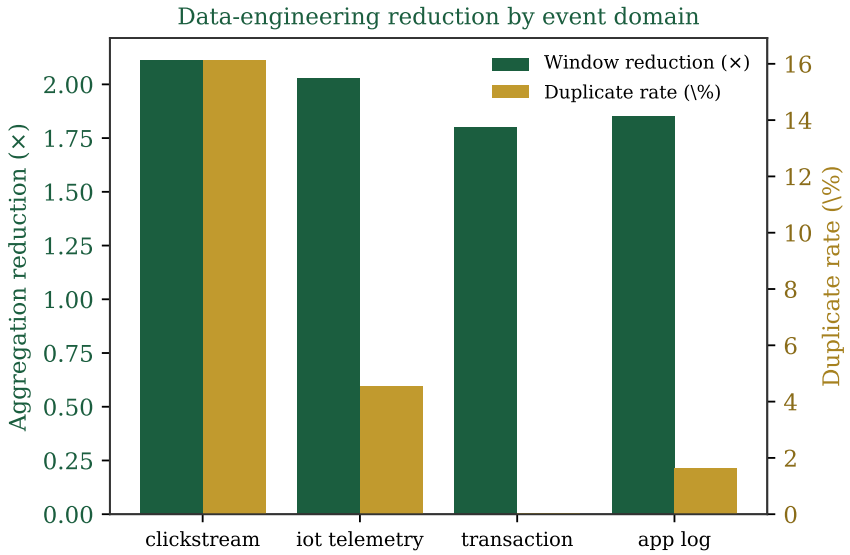

Figure 2. Window aggregation reduction ratio and duplicate rate by event domain. Domains with more repetition compress more, so the same operator yields different savings across the enterprise.

6.3. Sketch accuracy

Table 2 compares the HyperLogLog estimate of the total distinct entity count against the exact value of 68468, across register counts from 64 to 4096. The measured relative error falls from 11.68 percent at $m = 64$ to 0.81 percent at $m = 4096$, closely tracking the theoretical $1.04/\sqrt{m}$ prediction shown alongside it. Figure 3 plots the measured and theoretical error on logarithmic axes, where the two lines share the same slope, confirming the square root memory law in practice.

Table 2. HyperLogLog cardinality accuracy against an exact count of 68468.

Registers m	Estimate	Rel. error (%)	Theory $1.04/\sqrt{m}$ (%)	Memory (B)
64	76463	11.68	13.00	64
128	74910	9.41	9.19	128
256	71721	4.75	6.50	256
512	70808	3.42	4.60	512
1024	70651	3.19	3.25	1024
2048	69092	0.91	2.30	2048
4096	69023	0.81	1.62	4096

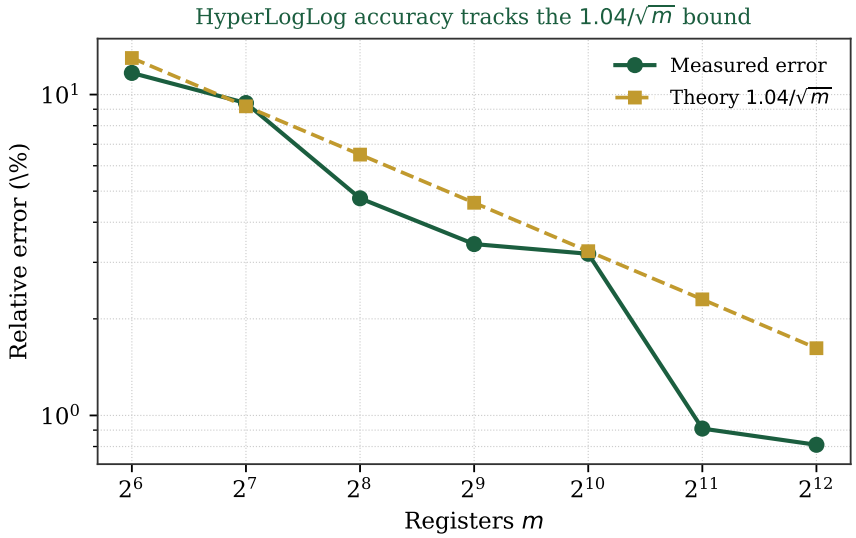

Figure 3. Measured HyperLogLog relative error against the theoretical $1.04/\sqrt{m}$ bound. Accuracy improves as the square root of the register memory, so a small fixed budget already yields single digit error.

For heavy hitters, Figure 4 shows the mean relative error over the true top fifty entities for a Count Min sketch of depth four as its width grows from 256 to 8192 columns. The error falls from 34.8 percent at the smallest width to 0.32 percent at the largest, matching the theory that the additive error scales inversely with the width. Reservoir sampling is equally well behaved. A reservoir of twenty thousand events reproduces the population mix of the four domains within 0.079 percentage points in the worst case, which is consistent with the uniformity proved above.

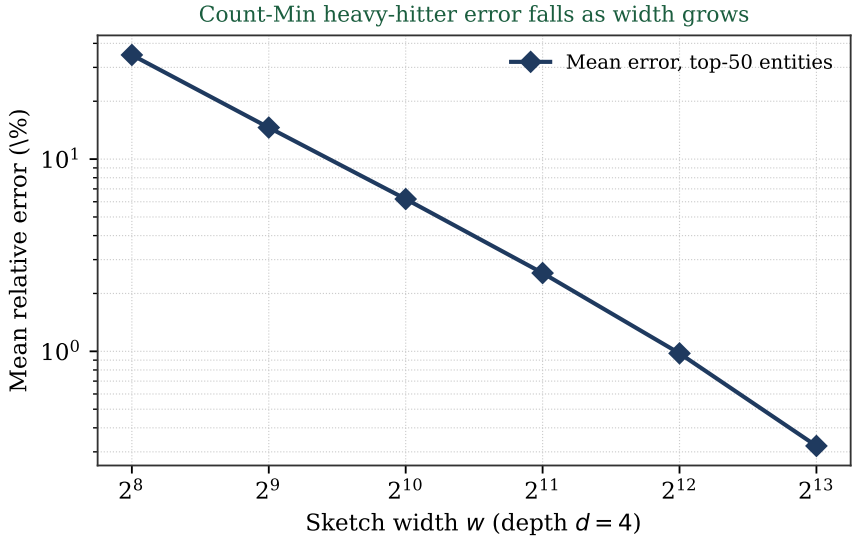

Figure 4. Count Min mean relative error over the true top fifty entities as the sketch width grows at fixed depth four. Wider sketches drive heavy hitter error below one percent.

6.4. Token cost for an event driven agent

Finally we measure the effect of reduction on the agent’s token bill. We take a slice of six thousand events, the kind of observation an agent would read in one tool round, and serialize it two ways: as raw JSON event lines and as a windowed aggregate summary carrying the same decision relevant signal of counts and per group statistics. The byte pair encoder yields 41901 tokens for the raw representation and 25536 tokens for the summary, a token reduction of $\eta = 39.1$ percent by Equation (2), while collapsing six thousand events into about three thousand summary rows. Substituting the resulting scale factor $\kappa = 0.61$ into the episode model (3) for an eight round agent with geometric

context growth, the cumulative cost drops from 322427 to 197522 tokens, a saving of 38.7 percent. Figure 5 shows the two trajectories diverging as rounds accumulate, which confirms that acting on the observation size is the highest leverage intervention for an event driven agent.

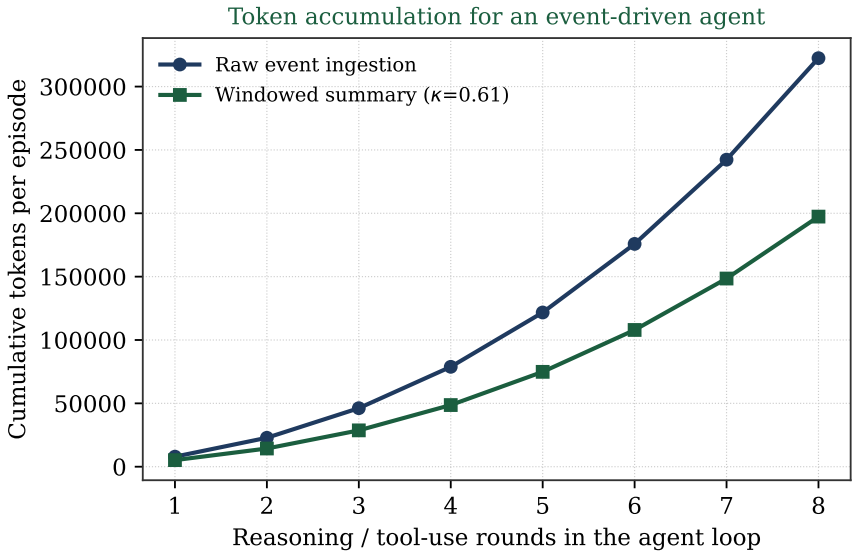

Figure 5. Cumulative tokens per episode against reasoning rounds for an event driven agent, feeding raw events versus windowed summaries. The gap widens geometrically, so data engineering reduction yields the largest per episode saving.

7. Discussion

The results connect two design surfaces that are usually tuned in isolation. On the data plane, window aggregation, cardinality and frequency sketches, and uniform sampling reduce a torrent of events to a compact, bounded artefact with provable properties: mergeability for parallelism, a square root memory law for distinct counts, an additive width controlled error for heavy hitters, and exact uniformity for samples. On the model plane, that same reduction is what sets the token size of every agent observation, and because episode cost grows geometrically in the number of rounds, a constant factor reduction in the observation compounds into a large absolute saving. The practical message is that the cheapest place to control an agent's cost is upstream, in the pipeline, long before a prompt is written.

Two cautions accompany the gains. First, aggressive reduction must preserve the signal the task relies on. A summary that keeps counts and per group statistics serves an agent that reasons about rates and totals, but an agent that must inspect a specific rare event needs the reservoir path or a targeted lookup, so the operators are complementary rather than substitutes. This is the same discipline of measuring before discarding that governs imbalanced learning on streaming data [27] and privacy aware publishing [24], where a careless transformation silently destroys the minority signal. Second, the reduced representation changes what the model can audit, so pipelines that must explain a decision should persist the pre reduction events in a transactional table for replay [14]. The mathematics of the reduction layer is also closely related to classical combinatorial and optimization structure that recurs across the author's prior work, from fuzzy graph colouring [29] to thermodynamically inspired search over hybrid landscapes [28].

The methods generalize beyond the four domains studied here. Applied deep learning systems for medical screening, including breast cancer prediction [30], brain tumour detection [31], and renal disease prediction [32], increasingly run as agents over streams of clinical events, and financial fraud detection [33] is inherently a heavy hitter and anomaly problem over transaction streams. Neuromorphic and edge deployments face the same pressure to represent information compactly, whether in real time image processing with generative models [34] or in home and industrial automation stacks [35]. Even sensing and Internet of Things applications, from smart waste management [36] and public safety [37] to smart home access control [38] and image enhancement pipelines [39], benefit when their agentic front ends spend tokens frugally over a reduced event stream.

8. Conclusion and Future Work

We presented a unified account of large scale event handling for token efficient agentic systems. By treating the data engineering pipeline as a composition of mergeable window operators and streaming sketches, we tied the reduction of a one million event stream directly to the token cost of the agent that consumes it. The reduction layer processes

millions of events per second, tracks its theoretical accuracy for distinct counts and heavy hitters, and samples without bias, and feeding the agent windowed summaries rather than raw events lowers the per episode token consumption by 38.7 percent for an eight round agent. Future work will extend the analysis to sliding and session windows with watermark driven triggers, to adaptive sketch sizing that spends memory where entity churn is highest, and to a closed loop in which the agent chooses the reduction granularity per round under an explicit token budget. We also plan to integrate the pipeline with retrieval augmented generation so that the same reduced artefacts serve both immediate reasoning and longer term memory.

Acknowledgements

The author thanks colleagues and collaborators for discussions on data engineering and agentic systems that informed this work.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017, pp. 5998–6008.

[2] T. Brown *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 1877–1901.

[3] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: synergizing reasoning and acting in language models," in *Proc. International Conference on Learning Representations (ICLR)*, 2023.

[4] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: language models can teach themselves to use tools," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023.

[5] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 9459–9474.

[6] P. Gage, "A new algorithm for data compression," *The C Users Journal*, vol. 12, no. 2, pp. 23–38, 1994.

[7] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proc. 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016, pp. 1715–1725.

[8] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[9] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache Spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[10] 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.

[11] J. Kreps, N. Narkhede, and J. Rao, "Kafka: a distributed messaging system for log processing," in *Proc. NetDB Workshop on Networking Meets Databases*, 2011, pp. 1–7.

[12] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle, "The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proc. VLDB Endowment*, vol. 8, no. 12, pp. 1792–1803, 2015.

[13] 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 (PODS)*, 2002, pp. 1–16.

[14] M. Armbrust *et al.*, "Delta Lake: high-performance ACID table storage over cloud object stores," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 3411–3424, 2020.

[15] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm," in *Proc. Conference on Analysis of Algorithms (AofA)*, 2007, pp. 127–146.

[16] 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.

[17] J. S. Vitter, "Random sampling with a reservoir," *ACM Transactions on Mathematical Software*, vol. 11, no. 1, pp. 37–57, 1985.

[18] G. Cormode, "Data sketching," *Communications of the ACM*, vol. 60, no. 9, pp. 48–55, 2017.

[19] L. Wang *et al.*, "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, art. 186345, 2024.

[20] S. Singamsetty and S. Sanakkayala, "Agentic AI-driven portfolio optimization: a hybrid approach for optimized stock selection and deep learning in algorithmic trading," *Multimedia Systems*, vol. 32, no. 1, art. 70, 2026.

[21] S. Satyanarayana and S. Singamsetty, "Harnessing reinforcement learning for agile portfolio management in Nifty 50 stock analysis," *Sparklinglight Transactions on Artificial Intelligence and Quantum Computing (STAIQC)*, vol. 4, no. 1, pp. 32–42, 2024.

[22] S. Satyanarayana, S. Kilaru, and K. Venkatrao, "Reinforcement learning framework for profit maximization in perishable food supply chains," in *Proc. 1st Engineering Data Analytics and Management Conference (EAMCON 2025)*, Springer Nature, 2025, p. 156.

[23] S. Singamsetty, S. Singamsetty, S. Satyanarayana, and P. R. Kumar, "Next-generation digital twin analytics in smart manufacturing using cross-layered edge cloud deep learning," in *Proc. 2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC)*, 2026, pp. 1–6.

[24] K. R. Rao, D. Nandan, and S. Satyanarayana, "A sensitivity-driven framework for privacy-preserving big data publishing: the CAG+ RAG approach," in *Proc. 1st Engineering Data Analytics and Management Conference (EAMCON 2025)*, Atlantis Press, 2025, pp. 189–198.

[25] P. S. Rao and S. Satyanarayana, "Privacy preserving data publishing based on sensitivity in context of big data using Hive," *Journal of Big Data*, vol. 5, no. 1, art. 20, 2018.

[26] S. Satyanarayana, Y. Tayar, and R. S. R. Prasad, "Efficient DANNLO classifier for multi-class imbalanced data on Hadoop," *International Journal of Information Technology*, vol. 11, no. 2, pp. 321–329, 2019.

[27] Y. Tayar, R. S. R. Prasad, and S. Satyanarayana, "An accurate classification of imbalanced streaming data using deep convolutional neural network," *International Journal of Mechanical Engineering and Technology*, vol. 9, no. 3, pp. 770–783, 2018.

[28] S. Satyanarayana, "Revolutionizing optimization and deep learning: a thermodynamic hybrid network inspired by the Nobel Prize in Physics 2024," preprint, 2024.

[29] S. Satyanarayana, "Range-valued fuzzy colouring of interval-valued fuzzy graphs," *Pacific Science Review A: Natural Science and Engineering*, vol. 18, no. 3, pp. 169–177, 2016.

[30] C. Mamatha, S. Satyanarayana, and T. K. Mohammad, "Enhanced automated system for early breast cancer prediction and classification using machine learning techniques," in *AIP Conference Proceedings*, vol. 3418, no. 1, AIP Publishing, 2026.

[31] A. Gupta, D. Nandan, S. Satyanarayana, and N. R. Rao, "Deep learning approach for brain glioblastoma detection: integrating efficient segmentation and survival rate prediction," in *Proc. 2026 IEEE International Conference for Convergence in Computing Technology (I3CTCON)*, 2026, pp. 1–7.

[32] S. Satyanarayana, T. Khatoon, and N. M. Bindu, "Breaking barriers in kidney disease detection: leveraging intelligent deep learning and artificial gorilla troops optimizer for accurate prediction," *International Journal of Applied and Natural Sciences*, vol. 1, no. 1, pp. 22–41, 2023.

[33] S. P. S. Appalabatla, A. S. Kumar, A. P. Sai, A. Sanjana, and S. Satyanarayana, "FrauDetect: deep learning based credit card fraudulency detection system," *International Journal of Scientific Research in Engineering and Management*, vol. 7, no. 6, 2023.

[34] N. M. Bindu and S. Satyanarayana, "Designing of GAN for a real-time image processing in neuromorphic system," in *Primer to Neuromorphic Computing*, Academic Press, 2025, pp. 21–44.

[35] S. Satyanarayana, T. Khatoon, and N. M. Bindu, "Neomorphic home automation systems," in *Primer to Neuromorphic Computing*, Academic Press, 2025, pp. 97–125.

[36] G. Vaisali, K. S. Bhargavi, S. Kumar, and S. Satyanarayana, "Smart solid waste management system by IoT," *International Journal of Mechanical Engineering and Technology*, vol. 8, no. 12, pp. 841–846, 2017.

[37] A. P. Begum, S. Satyanarayana, and K. Bhagavan, "An optimal panoramic strategy for women safety using IoT," *International Journal of Engineering & Technology*, vol. 7, no. 1.6, 2018.

[38] S. Marrapu, S. Satyanarayana, V. Arunkumar, and J. D. S. K. Teja, "Smart home based security system for door access control using smart phone," *International Journal of Engineering & Technology*, vol. 7, no. 1, p. 249, 2018.

[39] A. V. S. N. Murty, B. N. Jagadesh, K. Bhagavan, and S. Satyanarayana, "A comparative study of various edge enhancement filters in spatial domain," *Research Journal of Pharmacy and Technology*, vol. 9, no. 12, pp. 2403–2406, 2016.