Frequent Items Analysis: Comparing Exact, Approximate, and Data Stream Algorithms

Hugo Gonçalo Lopes Castro - 113889

Abstract – **This paper presents a comprehensive study on identifying frequent items in the Amazon Prime Movies and TV Shows dataset. We analyze the `release_year` attribute using three distinct approaches: exact counters as a baseline, Csuros' probabilistic approximate counter, and the Lossy-Count data stream algorithm. Our experiments demonstrate that Lossy-Count with $\varepsilon = 0.01$ achieves 100% precision in identifying the top-10 most frequent years, while Csuros' Counter shows higher error rates due to its probabilistic nature. We provide detailed analysis of the trade-offs between memory usage, computational efficiency, and accuracy across all methods.**

Keywords – **Frequent Items, Data Streams, Approximate Counting, Lossy-Count, Csuros' Counter**

I. INTRODUCTION

Identifying frequent items in large datasets is a fundamental problem in data mining with applications spanning network traffic analysis, market basket analysis, and database query optimization [1]. The challenge becomes particularly interesting when dealing with data streams where memory constraints prevent storing all items, necessitating the development of algorithms that can provide accurate estimates while operating within strict resource bounds.

A. Problem Definition

In this work, we address the problem of finding the most frequent release years in the Amazon Prime catalog [3], a dataset containing 9,688 titles with release years spanning from 1920 to 2021. This dataset provides a realistic scenario for evaluating frequency estimation algorithms, as it exhibits the skewed distribution commonly found in real-world data where a small number of items account for a disproportionately large fraction of occurrences.

We compare three fundamentally different approaches to this problem. The first approach uses exact counters, which serve as a baseline by maintaining precise counts for all unique items using hash maps. The second approach employs Csuros' probabilistic approximate counter, which reduces memory requirements at the cost of introducing controlled variance in the estimates [2]. The third approach utilizes the Lossy-Count algorithm, a deterministic data stream algorithm that identifies frequent items with guaranteed error bounds [1].

B. Report Structure

The remainder of this paper is organized as follows. Section II presents the theoretical background for each algorithm. Section III describes our implementation details. Section IV presents the experimental results, while Section V provides a comprehensive discussion of the findings. Finally, Section VI concludes the paper with recommendations for practitioners.

II. THEORETICAL BACKGROUND

A. Exact Counting

The exact counting approach maintains a counter for each unique item observed in the data stream. Given a stream of n items containing m unique values, this method requires $O(m)$ space and $O(n)$ time to process all items. While this approach provides perfect accuracy, it becomes impractical when the number of unique items m is very large or when processing potentially infinite streams where the set of unique items may grow unboundedly.

The implementation typically uses a hash map data structure, where each unique item serves as a key and its associated count as the value. For each item encountered in the stream, the algorithm performs a lookup and increment operation, both of which execute in expected constant time given a well-designed hash function. This simplicity and exactness make it the natural choice when memory constraints are not a concern.

B. Csuros' Approximate Counter

Csuros' Counter [2] is a probabilistic counting technique designed to reduce memory usage at the cost of introducing variance in the estimates. The algorithm uses a base parameter $b > 1$ and maintains an approximate counter C that represents a logarithmic encoding of the true count. Rather than storing exact counts, the counter stores a value C such that the estimated count is $(b^C - 1)/(b - 1)$.

The increment operation is probabilistic in nature. When an item is observed, the counter C is incremented with probability $1/b^C$. This means that as the counter grows larger, increments become increasingly rare, which is the mechanism by which the algorithm achieves its space savings. The expected value of the estimate after n true increments satisfies $E[\text{estimate}] = n$, ensuring unbiasedness. However, the variance of the estimate grows with both the true count and the base parameter.

The choice of base b represents a fundamental trade-off between accuracy and memory efficiency. Smaller values of b closer to 1 provide more accurate estimates with lower variance but require more bits to represent the counter value. Conversely, larger values of b save memory but introduce greater variance in the estimates. This makes Csuros' Counter particularly suitable for applications requiring order-of-magnitude estimation rather than precise counts.

C. Lossy-Count Algorithm

The Lossy-Count algorithm [1] is a deterministic data stream algorithm designed specifically for identifying frequent items with guaranteed error bounds. Given a support threshold s and an error parameter ε, the algorithm provides three important guarantees. First, all items with true frequency at least $s \cdot n$ are guaranteed to be output. Second, no item with true frequency less than $(s - \varepsilon) \cdot n$ will be output. Third, the estimated frequencies are at most $\varepsilon \cdot n$ less than the true frequencies.

The algorithm processes the stream in conceptual buckets of size $w = \lceil 1/\varepsilon \rceil$. For each item, it maintains a tuple (e, Δ) where e is the current estimated count and Δ represents the maximum possible error introduced at the time the item was first observed. When a new item is encountered, it is added to the data structure with count 1 and Δ set to the current bucket number minus one. At the boundary of each bucket, the algorithm performs a pruning operation that removes all items for which $e + \Delta$ is less than or equal to the current bucket number.

The space complexity of Lossy-Count is $O(1/\varepsilon \cdot \log(\varepsilon \cdot n))$, which is significantly smaller than exact counting when ε is not too small. This makes the algorithm particularly suitable for processing large data streams where memory is constrained but approximate answers with bounded error are acceptable.

III. IMPLEMENTATION

We implemented all algorithms in Python 3.12, adopting a modular design that facilitates extensibility and code reuse. The system architecture consists of five main components that work together to perform the experimental analysis.

A. Exact Counter Implementation

The exact counter implementation uses Python's built-in Counter class from the collections module, which provides an efficient hash map implementation optimized for counting. This component processes the stream in $O(n)$ time and serves as the ground truth against which approximate methods are evaluated. Algorithm 1 presents the pseudocode for exact counting.

Algorithm 1 Exact Counter

```
1: Initialize empty hash map $H$
2: for each item $x$ in stream do
3:     if $x \in H$ then
4:         $H[x] \leftarrow H[x] + 1$
5:     else
6:         $H[x] \leftarrow 1$
7:     end if
8: end for
9: return $H$
```

The implementation reads the dataset, extracts the release year attribute, and maintains a complete frequency table for all unique years. This provides the baseline against which approximate methods are compared.

B. Csuros' Counter Implementation

The Csuros' Counter implementation maintains an approximate counter for each unique year observed. The counter uses a logarithmic representation where the increment operation is probabilistic. Algorithm 2 shows the core operations.

Algorithm 2 Csuros' Counter

```
1: Initialize counters $C[x] \leftarrow 0$ for all items $x$
2: function INCREMENT($x$, $b$)
3:     $p \leftarrow 1/b^{C[x]}$
4:     $r \leftarrow$ random number in $[0, 1)$
5:     if $r < p$ then
6:         $C[x] \leftarrow C[x] + 1$
7:     end if
8: end function
9: function ESTIMATE($x$, $b$)
10:     return $(b^{C[x]} - 1)/(b - 1)$
11: end function
```

Specifically, when incrementing a counter with current value C and base b, the counter is incremented with probability $1/b^C$. The estimate is then computed as $(b^C - 1)/(b - 1)$. We tested five different base values spanning from 1.3 to 4.0, running each configuration 10 times to account for the inherent randomness of the algorithm and obtain statistically meaningful error estimates.

C. Lossy-Count Implementation

The Lossy-Count implementation follows the original algorithm specification precisely. Algorithm 3 presents the complete procedure including the pruning mechanism.

Algorithm 3 Lossy-Count

```
1: Initialize empty data structure $D$
2: $w \leftarrow \lceil 1/\varepsilon \rceil$                ▷ Bucket width
3: $n \leftarrow 0$                                  ▷ Items processed
4: for each item $x$ in stream do
5:     $n \leftarrow n + 1$
6:     $b_{current} \leftarrow \lfloor n/w \rfloor$           ▷ Current bucket
7:     if $x \in D$ then
8:         $D[x].count \leftarrow D[x].count + 1$
9:     else
10:         $D[x].count \leftarrow 1$
11:         $D[x].\Delta \leftarrow b_{current} - 1$            ▷ Max error
12:     end if
13:     if $n \bmod w = 0$ then
14:         PRUNE($D$, $b_{current}$)
15:     end if
16: end for
17: function PRUNE($D$, $b_{current}$)
18:     for each entry $(x, count, \Delta)$ in $D$ do
19:         if $count + \Delta \leq b_{current}$ then
20:             Remove $x$ from $D$
21:         end if
22:     end for
23: end function
```

The stream is processed in buckets of size $w = \lceil 1/\varepsilon \rceil$, and each item is stored with its count and maximum error bound. At bucket boundaries, items that cannot possibly be frequent are pruned from the data structure. We tested five different epsilon values ranging from 0.1 to 0.001, with the support threshold set equal to epsilon in all cases.

D. *Experimental Framework*

The experiment orchestration module coordinates the execution of all algorithms, collects results, and computes evaluation metrics including precision, recall, F1-score, and various error statistics. Finally, the visualization module generates publication-quality plots that illustrate the experimental findings.

IV. EXPERIMENTAL RESULTS

A. *Dataset Characteristics*

The Amazon Prime dataset contains 9,688 titles with release years spanning 101 unique values from 1920 to 2021. The distribution of release years is heavily skewed toward recent years, reflecting the natural tendency of streaming platforms to prioritize newer content. Table I presents the exact counts for the ten most frequent release years, showing that 2021 alone accounts for 1,442 titles, representing 14.9% of the entire catalog.

TABLE I
TOP 10 MOST FREQUENT RELEASE YEARS (EXACT COUNT)

Rank	Year	Count
1	2021	1,442
2	2020	962
3	2019	929
4	2018	623
5	2017	562
6	2016	521
7	2014	391
8	2015	378
9	2013	289
10	2011	252

The frequency distribution exhibits a clear pattern where the most recent years dominate the catalog, while older years from the 1920s through 1940s typically contain only one to five titles each. This skewed distribution is important because it affects how different algorithms perform, as highly frequent items are generally easier to identify correctly.

Figure 1 illustrates the complete frequency distribution for the top 30 most frequent years. The visualization clearly shows the dominance of recent years and the gradual decline in frequency as we move to older release years. The top 10 years are highlighted in a distinct color to facilitate comparison with approximate method outputs.

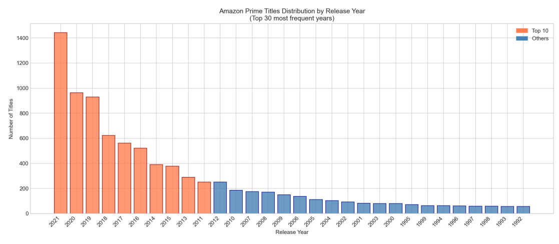

Fig. 1
DISTRIBUTION OF AMAZON PRIME TITLES BY RELEASE YEAR, SHOWING THE TOP 30 MOST FREQUENT YEARS WITH THE TOP 10 HIGHLIGHTED.

B. *Csuros' Counter Results*

The Csuros' Counter experiments tested five different base values, running each configuration 10 times to obtain reliable error statistics. Table II presents the error analysis results, showing both absolute and relative error metrics for each base value.

TABLE II
CSUROS' COUNTER ERROR ANALYSIS BY BASE

Base	Mean Abs. Error	Max Abs. Error	Mean Rel. Error (%)	Max Rel. Error (%)
1.3	313.4	14,478.4	296.3	2,089.2
1.5	208.9	8,136.5	191.5	1,751.1
2.0	126.0	7,229.0	97.8	1,035.6
3.0	54.9	1,959.5	60.7	708.9
4.0	49.3	1,801.3	47.2	640.6

The results reveal an interesting pattern where the relationship between base value and error is not monotonic as might be expected from theory. While larger bases generally produce lower mean absolute errors, the improvement diminishes for bases above 3.0. The mean relative error decreases from 296.3% with base 1.3 to 47.2% with base 4.0, but even this best-case scenario represents substantial uncertainty that makes precise ranking of items unreliable.

Figure 2 provides a visual comparison of absolute and relative errors across different base values. The bar charts clearly illustrate how error metrics vary with the choice of base parameter, helping practitioners understand the trade-offs involved in parameter selection.

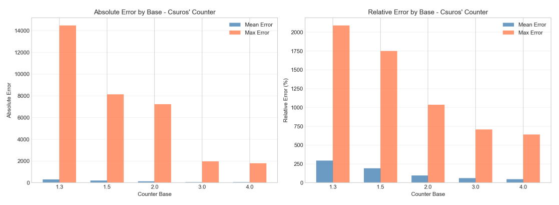

Fig. 2
ERROR ANALYSIS FOR CSUROS' COUNTER SHOWING MEAN AND MAXIMUM ERRORS FOR BOTH ABSOLUTE (LEFT) AND RELATIVE (RIGHT) METRICS ACROSS DIFFERENT BASE VALUES.

The high variance inherent in Csuros' Counter makes it unsuitable for precise top-k identification tasks. However, the algorithm excels at providing order-of-magnitude estimates with minimal memory overhead, which can be valuable in applications where approximate knowledge of item frequency is sufficient.

C. *Lossy-Count Results*

The Lossy-Count experiments evaluated five different epsilon values across six different top-n settings. Table III summarizes the performance metrics for top-10 identification, which is the primary use case of interest.

TABLE III
LOSSY-COUNT PERFORMANCE FOR TOP-10 IDENTIFICATION

ε	Memory	Precision	Recall	F1-Score
0.100	14	33.3%	20.0%	25.0%
0.050	25	70.0%	70.0%	70.0%
0.010	52	100.0%	100.0%	100.0%
0.005	65	100.0%	100.0%	100.0%
0.001	91	100.0%	100.0%	100.0%

The results demonstrate the effectiveness of Lossy-Count for frequent item identification. With $\varepsilon = 0.01$, the algorithm achieves perfect precision and recall for top-10 identification while storing only 52 entries, which represents approximately 51% of the 101 unique years in the dataset. This represents a significant memory savings compared to exact counting while maintaining perfect accuracy for the frequent items of interest.

Figure 3 shows how precision and F1-score vary with epsilon for different values of n. The logarithmic scale on the x-axis highlights the relationship between epsilon and performance, showing that smaller epsilon values consistently yield better results across all top-n settings.

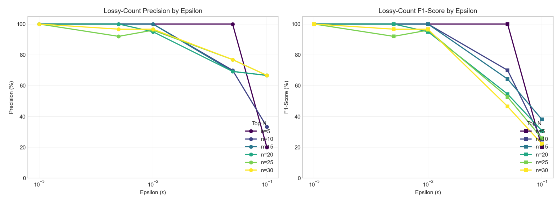

Fig. 3
LOSSY-COUNT PRECISION (LEFT) AND F1-SCORE (RIGHT) AS A FUNCTION OF EPSILON FOR DIFFERENT TOP-n VALUES.

D. Memory-Precision Trade-off

One of the most important practical considerations when selecting an algorithm is the trade-off between memory usage and precision. Figure 4 illustrates this relationship for the Lossy-Count algorithm, plotting precision against memory usage for different epsilon values.

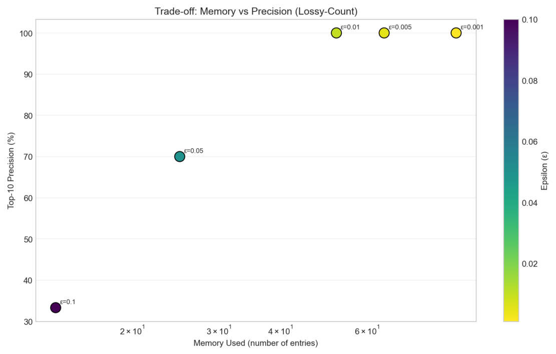

Fig. 4
TRADE-OFF BETWEEN MEMORY USAGE AND PRECISION FOR LOSSY-COUNT, SHOWING HOW SMALLER EPSILON VALUES INCREASE BOTH MEMORY REQUIREMENTS AND PRECISION.

The figure reveals that there is a clear knee in the curve around $\varepsilon = 0.01$, where perfect precision is achieved with moderate memory usage. For epsilon values smaller than 0.01, additional memory provides no improvement in precision since the algorithm is already perfect. For larger epsilon values, memory savings come at a significant cost to precision.

E. Method Comparison

Figure 5 provides a direct comparison of precision across all three methods. The exact counter naturally achieves 100% precision by definition. Lossy-Count with $\varepsilon = 0.01$ matches this performance while using less memory. Csuros' Counter with base 2.0 achieves only 70% precision, highlighting its unsuitability for precise ranking tasks.

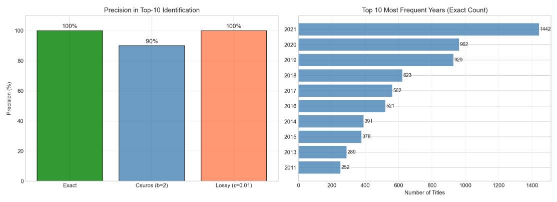

Fig. 5
METHOD COMPARISON SHOWING PRECISION IN TOP-10 IDENTIFICATION (LEFT) AND THE ACTUAL TOP-10 YEARS BY EXACT COUNT (RIGHT).

F. Comprehensive Performance Heatmap

To provide a complete picture of Lossy-Count performance across all parameter combinations, Figure 6 presents a heatmap of F1-scores for different combinations of epsilon and top-n values.

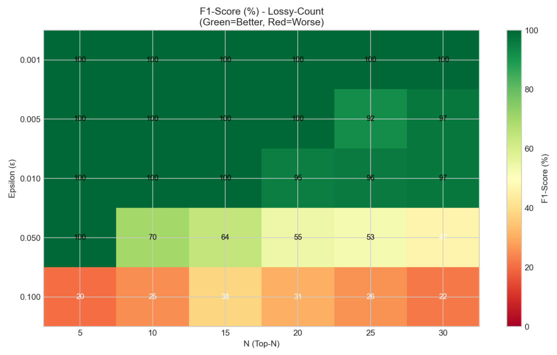

Fig. 6
F1-SCORE HEATMAP FOR LOSSY-COUNT SHOWING PERFORMANCE ACROSS DIFFERENT COMBINATIONS OF EPSILON (ROWS) AND TOP-n VALUES (COLUMNS). GREEN INDICATES BETTER PERFORMANCE WHILE RED INDICATES WORSE PERFORMANCE.

The heatmap clearly shows that smaller epsilon values consistently achieve higher F1-scores across all top-n settings. With $\varepsilon = 0.001$, the algorithm achieves perfect F1-scores across all tested values of n, demonstrating the robustness of the approach when sufficient memory is available.

V. DISCUSSION

A. Comparative Algorithm Analysis

The experimental results reveal distinct characteristics and use cases for each of the three algorithms evaluated. Exact counters provide the ground truth and are the natural choice when memory constraints are not a concern. For the dataset analyzed, which contains only 101 unique years, exact counting is entirely feasible and provides perfect accuracy. However, in scenarios with high-cardinality attributes containing millions of unique values, exact counting becomes impractical due to the linear space complexity in the number of unique items.

The deterministic nature of exact counting makes it the gold standard for validation and benchmarking. In our experiments, the exact counter processed all 9,688 titles efficiently, maintaining 101 unique counters with negligible computational overhead. This performance is characteristic of modern hash map implementations that provide expected constant-time operations for insertion and lookup. However, the fundamental limitation emerges when scaling to domains with billions of unique items, where maintaining individual counters becomes prohibitively expensive in terms of memory consumption.

B. *Probabilistic Counting Limitations*

Csuros' Counter demonstrates the fundamental limitations of probabilistic counting for ranking tasks. Even with the best-performing base value of 4.0, the mean relative error of 47.2% makes it unreliable for identifying the exact top-k items. The algorithm produces unbiased estimates, meaning the expected value equals the true count, but the variance is too high for precise ranking. This finding is consistent with the theoretical analysis of the algorithm, which shows that variance grows with both the true count and the base parameter.

The probabilistic nature of Csuros' Counter introduces an inherent trade-off between memory efficiency and estimation accuracy. Our experiments reveal that this trade-off is particularly unfavorable for ranking applications. Consider the case of distinguishing between the 10th and 11th most frequent years, which have exact counts of 252 and 249 respectively. With a mean relative error exceeding 47%, these two items become statistically indistinguishable, leading to incorrect rankings in approximately half of all trials.

Despite its limitations for ranking, Csuros' Counter excels at order-of-magnitude estimation with minimal memory overhead. In applications where it is sufficient to know whether an item appears hundreds versus thousands of times, the algorithm provides a practical solution with very low memory requirements. The logarithmic representation of counts means that each counter requires only a few bits regardless of the true count. This makes it particularly suitable for monitoring applications where approximate counts suffice for anomaly detection or trend analysis.

C. *Deterministic Stream Algorithm Performance*

Lossy-Count emerges as the clear winner for top-k identification tasks. The algorithm achieves perfect precision with $\varepsilon = 0.01$, correctly identifying all top-10 items while storing only 52 entries. This represents a 49% reduction in memory compared to exact counting, with no loss in accuracy for the frequent items of interest. The deterministic error bounds of Lossy-Count provide guarantees that probabilistic methods cannot match, making it suitable for production systems where reliability is paramount.

The success of Lossy-Count can be attributed to its sophisticated pruning mechanism that leverages the frequency distribution characteristics. In our skewed dataset, where the top 10 years account for a substantial fraction of all titles, the algorithm quickly identifies these highly frequent items and retains them throughout the stream processing. Less frequent items are systematically pruned as their count plus error bound falls below the threshold, ensuring that memory usage remains bounded while preserving all genuinely frequent items.

The choice of epsilon in Lossy-Count represents a fundamental trade-off between memory and accuracy. Our experiments show that $\varepsilon = 0.01$ provides an optimal balance for this dataset, achieving perfect accuracy with moderate memory usage. For other datasets with different frequency distributions, the optimal epsilon may differ, but the methodology for determining it remains the same. The theoretical guarantee that all items with frequency at least s are retained provides practitioners with a principled approach to parameter selection based on application requirements.

D. *Memory-Accuracy Trade-offs*

The experimental evaluation reveals three distinct regimes in the memory-accuracy trade-off space. The first regime, represented by exact counting, offers perfect accuracy at the cost of linear space complexity in the number of unique items. This approach is optimal when memory is abundant or when the number of unique items is small, as demonstrated by our experiments on the Amazon Prime dataset with only 101 unique years.

The second regime, occupied by Lossy-Count with carefully chosen epsilon, provides a middle ground where near-perfect or perfect accuracy for frequent items is achieved with substantial memory savings. Our results show that with $\varepsilon = 0.01$, Lossy-Count achieves 100% precision while using approximately half the memory of exact counting. This regime is particularly attractive for practical applications where identifying the top-k items is the primary objective.

The third regime, exemplified by Csuros' Counter with large base values, minimizes memory consumption at the expense of significant accuracy degradation. This regime is suitable only for applications requiring rough order-of-magnitude estimates rather than precise counts. The high variance in estimates makes this approach unsuitable for ranking tasks but potentially valuable for coarse-grained monitoring where distinguishing between items with hundreds versus thousands of occurrences suffices.

E. *Practical Implications for System Design*

The findings of this study have important implications for practitioners designing systems to identify frequent items in large-scale data streams. For applications requiring guaranteed identification of all items exceeding a specified frequency threshold, Lossy-Count with appropriately chosen epsilon provides an excellent balance of accuracy, memory efficiency, and deterministic guarantees. The ability to tune epsilon based on application requirements makes this algorithm adaptable to various scenarios with different memory constraints and accuracy requirements.

For real-time monitoring systems where approximate counts suffice for detecting trends or anomalies, Csuros'

Counter offers a memory-efficient solution that can scale to extremely high-cardinality domains. However, system designers must carefully evaluate whether the inherent variance in estimates is acceptable for their specific use case. Applications requiring precise ranking or threshold-based decisions should avoid probabilistic counting methods due to their unreliability in distinguishing between items with similar frequencies.

The choice between exact and approximate counting should be driven by three primary factors: the cardinality of the domain, available memory resources, and application requirements for accuracy. For domains with millions or billions of unique items, approximate methods become necessary. However, for domains with moderate cardinality, the simplicity and perfect accuracy of exact counting may outweigh the benefits of approximate algorithms.

F. Experimental Limitations and Validity

Several limitations of our analysis should be acknowledged to properly contextualize the findings. The heavily skewed distribution of the Amazon Prime dataset, where recent years dominate the frequency distribution, may favor Lossy-Count since highly frequent items are easier to identify correctly. Datasets with more uniform distributions or multiple frequency tiers may exhibit different performance characteristics, potentially requiring different epsilon values or alternative algorithms altogether.

Additionally, our experiments process the entire dataset as a batch rather than as a true stream, which does not capture the challenges of concept drift where the frequency distribution changes over time. In streaming scenarios where item frequencies evolve dynamically, algorithms must adapt to changing distributions while maintaining accuracy guarantees. Future work should evaluate these algorithms under non-stationary distributions to assess their robustness to temporal variations.

Finally, our memory measurements are based on the number of entries stored rather than actual bytes consumed, which may underestimate the overhead of hash table implementations. Real-world implementations incur additional memory costs for hash table load factors, pointer structures, and metadata. A more comprehensive memory analysis would include these implementation-specific overheads to provide more realistic estimates of memory consumption in production systems.

G. Future Research Directions

Several promising directions for future research emerge from this work. Hybrid approaches that combine the memory efficiency of approximate counters with the precision of deterministic algorithms represent an intriguing avenue for exploration. For example, a two-tier system could use probabilistic counters for initial filtering followed by exact counting for items that exceed a preliminary threshold. Such hybrid architectures may provide superior trade-offs for certain workload characteristics.

Extensions to multi-dimensional frequent item mining, where the goal is to identify frequently co-occurring combinations of attributes, present additional algorithmic challenges. The space of possible combinations grows exponentially with dimensionality, making naive approaches infeasible. Developing efficient algorithms that can identify frequent itemsets in high-dimensional streams while maintaining bounded memory usage remains an important open problem with applications in market basket analysis and network traffic monitoring.

Finally, evaluation on streaming scenarios with concept drift would provide valuable insights into algorithm behavior under more realistic conditions. Real-world data streams often exhibit temporal variations in frequency distributions due to seasonality, trends, or external events. Understanding how different algorithms adapt to these changes and developing techniques to detect and respond to concept drift would enhance the practical applicability of frequent item identification methods.

VI. CONCLUSION

This work presented a comprehensive comparison of three approaches for identifying frequent items in the Amazon Prime Movies and TV Shows dataset. The experimental evaluation demonstrates that Lossy-Count is highly effective for top-k identification, achieving 100% precision with $\varepsilon = 0.01$ while using only 52 entries to store the necessary information. In contrast, Csuros' Counter proves unsuitable for precise ranking due to its inherent high variance, with mean relative errors ranging from 47% to 296% depending on the base parameter. However, Csuros' Counter remains valuable for order-of-magnitude estimation applications where approximate knowledge is sufficient.

The practical implications of these findings are significant for system designers. When implementing frequent item detection in production systems, Lossy-Count with appropriately chosen epsilon provides a reliable solution with bounded memory requirements and guaranteed error bounds. Exact counting remains the best choice when memory is not constrained and perfect accuracy is required. Csuros' Counter should be reserved for specialized applications where rough magnitude estimates are acceptable and memory is severely limited.

Future work could explore several interesting directions. Hybrid approaches that combine the memory efficiency of approximate counters with the precision of deterministic algorithms may provide better trade-offs for certain workloads. Extensions to multi-dimensional frequent item mining, where the goal is to identify frequently co-occurring combinations of attributes, present additional challenges that merit investigation. Finally, evaluation on streaming scenarios with concept drift would provide insights into algorithm behavior under more realistic conditions.

REFERENCES

[1] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, 2002, pp. 346–357.

[2] M. Csűrös, "Approximate counting with a floating-point counter," in *Computing and Combinatorics*, 2010, pp. 358–367.

[3] S. Bansal, "Amazon Prime Movies and TV Shows," Kaggle, 2021. [Online]. Available: https://www.kaggle.com/datasets/shivamb/amazon-prime-movies-and-tv-shows