AGILE: GIScience Series, 7, 31, 2026. https://doi.org/10.5194/agile-giss-7-31-2026
Proceedings of the 29th AGILE Conference on Geographic Information Science, 16–19 June 2026.
Eds.: Evelyn Uuemaa, Alexander Kmoch, Holger Virro, and Simon Scheider
This contribution underwent peer review based on a full paper submission.

CascadeCBF: Probabilistic Counting for Sparse Spatial Point Clouds

Moritz Laass [1], Paul Walther [1], Wejdene Mansour [1], and Martin Werner [1]

[1]TUM School of Engineering and Design, Technical University of Munich, Munich, Germany

Correspondence: Moritz Laass (moritz.laass@tum.de)

Abstract. Spatial point datasets often exhibit long-tail distributions, where few locations receive most observations. Traditional counting Bloom filters struggle with such distributions due to fixed counter bit-depths. We present CascadeCBF, a multi-layer probabilistic data structure using cascading overflow: counters in lower layers handle infrequent locations, while overflow cascades to higher layers for hotspots. CascadeCBF supports spatial operations (union, intersection, aggregation) directly on the compressed representation. Evaluation on Zipfian distributions shows that CascadeCBF with minimal increment matches the memory efficiency of Spectral Bloom Filters and Count-Min Sketch, while the multi-layer design enables high-precision counting for highly skewed data with a 1.3-1.6× higher throughput.

Submission Type. Short Paper **BoK Concepts.** [AM8] Geostatistics, [DM1-4] Data Structures and Indices for Databases, [DM1-5] Data compression techniques

Keywords. probabilistic data structures, counting Bloom filter, spatial data, cardinality estimation

1 Introduction

Geospatial applications generate point data at massive scales, billions of geotagged posts, GPS traces, sensor observations, and mobility records. A key operation is *cardinality estimation*, which counts observations per location for hotspot detection, trajectory analysis, and spatial aggregation. The distribution of points in space is not uniform: in urban traffic monitoring, intersections log millions of vehicles yearly, while suburban roads see dozens; in pandemic surveillance, cases cluster in population centers, while large areas of the world are unpopulated. These applications require counting structures that adapt to wildly varying frequencies across space.

Probabilistic structures like Counting Bloom Filters (CBFs) (Fan et al., 2000) offer space-efficient frequency estimation, but fixed-bit-depth counters either overflow at hotspots or waste memory at rare locations. This is problematic for spatial data, which typically follows power-law distributions: a few locations (city centers, popular venues) receive orders of magnitude more observations than the majority. A 4-bit counter overflows at count 15–insufficient for any urban hotspot–while 32-bit counters waste 28 bits for locations with fewer than 16 observations.

We introduce *CascadeCBF*, extending GloBiMaps (Werner, 2019) with a multi-layer counting Bloom filter with cascading overflow: layers use increasing bit-depths, and increments cascade upward when counters saturate. Unlike traditional multi-layer approaches that incur rebuild overhead or require Huffman coding, CascadeCBF uses a simple overflow bit protocol: each counter reserves its high bit as a flag, enabling $\mathcal{O}(1)$ overflow detection. This design achieves three properties simultaneously: (1) *cache efficiency*–most operations touch only the first layer; (2) *large count range*–two layers handle counts up to 2^{30}; (3) *composability*–spatial operations work directly on the compressed representation.

Our contributions are: (1) an overflow bit protocol for efficient multi-layer counting with cache-friendly access patterns; (2) spatial set operations (union, intersection, aggregation) directly on compressed representations; (3) comprehensive evaluation on four real-world spatial datasets demonstrating competitive accuracy and 1.3–1.6× throughput improvements over alternatives.

2 Background and Related Work

To clarify the background of our work, we first provide an introduction to Bloom filter theory, then discuss Bloom filter variants for counting and typical spatial data characteristics.

2.1 Bloom Filter Theory

A Bloom filter (Bloom, 1970) represents a set using an m-bit array and k hash functions h. Inserting element e sets bits at positions $h_1(e), \ldots, h_k(e)$; membership queries check if all k positions contain ones. The *fraction of zeros* after inserting n elements is:

$$\text{foz} = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m} \tag{1}$$

The false positive probability–finding k ones at random positions–is $p = (1 - \text{foz})^k$. Minimizing p yields the optimal number of hash functions: $k^* = (m/n)\ln 2 \approx 0.693(m/n)$.

2.2 Counting Extensions

Counting Bloom Filters (CBFs) (Fan et al., 2000) replace the single bits in each Bloom filter slot with b-bit counters, enabling frequency queries and deletions. However, fixed bit-depth creates a tradeoff: small counters overflow on hotspots while large counters waste memory on rare items. With $b = 4$ bits, counters overflow at 15; doubling to $b = 8$ increases memory $2\times$ while in uneven set distributions most counters never exceed single digits. This inefficiency compounds for spatial data where frequency variation spans orders of magnitude.

Spectral Bloom Filters (Cohen and Matias, 2003) improve accuracy via *minimal increment* (MI)–only incrementing counters at the current minimum count, reducing overcounting from collisions. Count-Min Sketch (Cormode and Muthukrishnan, 2005) provides provable error bounds.

Multi-layer approaches address the bit-depth tradeoff. Dynamic Bloom Filters (Guo et al., 2009) and Dynamic Count Filters (Aguilar-Saborit et al., 2006) resize dynamically but incur rebuild overhead when many counters overflow simultaneously. Multilayer Compressed CBFs (Ficara et al., 2008) use Huffman coding for variable-length counters, trading CPU cycles for memory efficiency. d-Left Counting Bloom Filters (Bonomi et al., 2006) use d-left hashing with multiple subtables, achieving near-optimal space at the cost of more complex insertion logic. In contrast, CascadeCBF uses a simple overflow bit protocol that requires no rebuild and no variable-length encoding: each counter reserves one bit as an overflow flag, enabling $\mathcal{O}(1)$ overflow detection.

2.3 Spatial Data Characteristics

Real-world spatial data follows power-law (Zipfian) distributions (Broder and Mitzenmacher, 2004): a few locations receive most observations, making fixed-sized counters particularly inefficient. The Zipfian distribution with parameter α assigns probability $p(r) \propto r^{-\alpha}$ to the r-th ranked item. Human mobility data typically exhibits $\alpha \in [1.0, 2.0]$: city centers, transit hubs, and popular venues dominate while suburban and rural areas remain sparse.

Bloom filters can be adapted for sparse spatial rasters by hashing discretized coordinates, supporting set operations (union, intersection) directly on the compressed representation (Werner, 2019). Coordinates are discretized via floor division: for resolution r, point (x, y) maps to cell $(\lfloor x/r \rfloor, \lfloor y/r \rfloor)$. This enables multiresolution queries–coarse queries use larger r for overview, fine queries use smaller r for detail.

3 CascadeCBF

CascadeCBF consists of k hash functions and layers $L_0, \ldots, L_{\Lambda-1}$ with increasing bit-depths $b_0 < \cdots < b_{\Lambda-1}$. The *overflow bit protocol* (Fig. 1) reserves each counter's high bit as overflow flag: when value bits saturate, the flag is set, and increments cascade to the next layer. Since Layer 1 is only accessed every 2^{b_0-1} increments, most operations touch only Layer 0–improving cache locality.

We use the *hashing trick* (Kirsch and Mitzenmacher, 2006): a single MurmurHash3 call produces two 64-bit values (h_1, h_2), from which k hash positions are derived as $j_i = (h_1 + i \cdot h_2) \bmod m_x, i \in [1, k]$. This reduces hash computation from $\mathcal{O}(k)$ to $\mathcal{O}(1)$ while maintaining the pairwise independence required for Bloom filter guarantees.

PUT and GET operations are $\mathcal{O}(k \cdot \Lambda)$ worst-case but typically $\mathcal{O}(k)$ since most counters remain in Layer 0. The 128-bit MurmurHash3 output provides excellent distribution for spatial coordinates, which often exhibit spatial autocorrelation that could defeat weaker hash functions.

Layer Sizing. Layers use asymmetric sizing: Layer 0 is large (handling the common case of low counts) while upper layers are progressively smaller. For n unique items with expected maximum count $c_{\max}$, Layer 0 should have $m_0 \geq k \cdot n / \ln 2$ counters for optimal Bloom filter performance. Layer 1 only stores overflow from the counters that exceed Layer 0 capacity. This asymmetry saves significant memory compared to uniform-sized layers. When $m_1 < m_0$, the same hash values map to a compressed index space in Layer 1. Since overflow events are rare for power-law data (typically $\beta = 1 - 5\%$), any resulting collisions contribute to the standard false positive rate. To keep a constant amount of hash functions through the cascading layers the Bloom filter size $m_1 = \beta m_0$, e.g., $m_1 = m_0/64$ for 1.5% overflow.

Modes. For the CascadeCBF, we propose two modes: the standard mode and an optional *minimal increment* (MI) mode, which only increments positions at minimum count, reducing false positive accumulation. Without MI, each insertion increments all k counters; collisions with other items cause overcounting proportional to filter load.

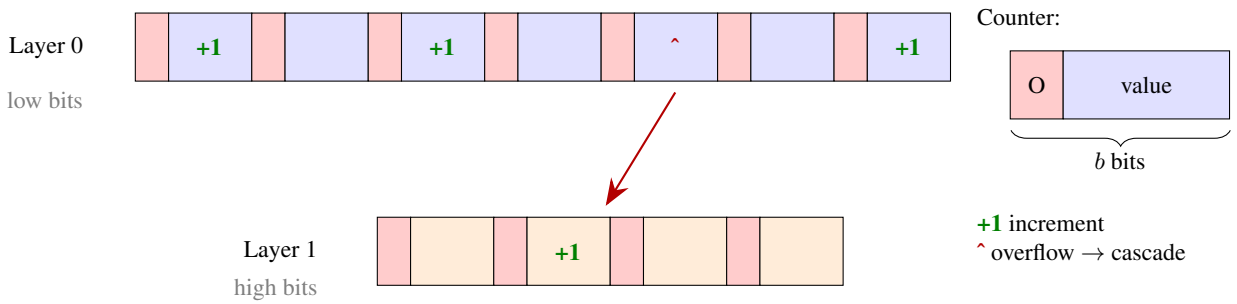

Figure 1. CascadeCBF overflow bit protocol. Layer 0 counters hold low-order bits; when value bits saturate (^), overflow bit O is set, value wraps to zero, and Layer 1 is incremented. Reconstruction sums contributions: count $= v_0 + v_1 \cdot 2^{b_0-1}$.

Algorithm 1 PUT Operation

Require: CascadeCBF A with layers L_x, point p
$h_1, h_2 \leftarrow$ **hash**(p)
for $i = 0$ **to** $k-1$ **do**
  for $x = 0$ **to** $\Lambda - 1$ **do**
    $j \leftarrow (h_1 + (i+1) \cdot h_2) \bmod m_x$
    if overflow bit at **val**(j, L_x) not set **then**
      increment **val**(j, L_x)
      break {move to next hash function}
    end if
  end for
end for

Algorithm 2 GET Operation

Require: CascadeCBF A, point p
Ensure: Estimated count $v_{\min}$
$h_1, h_2 \leftarrow$ **hash**(p); $v_{\min} \leftarrow \infty$
for $i = 0$ **to** $k-1$ **do**
  $s \leftarrow 0$
  for $x = 0$ **to** $\Lambda - 1$ **do**
    $j \leftarrow (h_1 + (i+1) \cdot h_2) \bmod m_x$
    $v \leftarrow$ **val**(j, L_x)
    if $v = 0$ **and** overflow bit not set **then**
      break {early exit: not present}
    end if
    $s \leftarrow s + v \cdot 2^{\sigma_x}$ $\{\sigma_0 = 0; \sigma_x = \sum_{i<x}(b_i - 1)\}$
    if overflow bit not set **then**
      break {no higher layers needed}
    end if
  end for
  $v_{\min} \leftarrow \min(v_{\min}, s)$
end for
return $v_{\min}$

With MI, only minimum-value counters are incremented, so overcounting only occurs when all k positions collide with items of equal or higher count–a rare event for high-frequency items. While standard mode provides 50% faster insertions when mild overcounting is tolerable, and supports element deletion via cascade decrement, we recommend MI mode for accuracy-critical applications; all evaluation tables use MI mode.

Error. The error of CascadeCBFs can be described by the overcount. This is the increase in the item count due to hash collisions. The expected total overcount for t insertions into n unique positions follows from standard Bloom filter analysis (Broder and Mitzenmacher, 2004):

$$\text{err}_{\text{total}} \approx k \cdot \frac{t}{n} \cdot \left(1 - e^{-kn/m}\right) \tag{2}$$

Data and Software Availability

The implementation of CascadeCBF is available as a header-only C++ library with python bindings (https://github.com/mlaass/cascade-cbf). GDELT event data is publicly available from the GDELT Project (Leetaru and Schrodt, 2013). COVID-19 case data was derived from the Johns Hopkins CSSE repository (Dong et al., 2020). NYC yellow taxi trip data is publicly available from the NYC Taxi and Limousine Commission (NYC Taxi and Limousine Commission, 2024).

4 Spatial Operations

A key advantage of CascadeCBF over general-purpose sketches is native support for spatial operations directly on compressed representations, avoiding decompression overhead.

Union. Merging two CascadeCBFs representing disjoint spatial regions requires element-wise counter addition. For each position j across all layers, we add corresponding counters: $A[j] = A_1[j] + A_2[j]$. The overflow bit protocol automatically handles cases where addition causes overflow–if the sum exceeds the layer capacity, the overflow bit is set and the carry propagates to the next layer. This enables distributed counting: partition space into tiles, maintain per-tile CascadeCBFs, and merge on demand.

Intersection. Computing the minimum count between two filters uses element-wise min: $A[j] = \min(A_1[j], A_2[j])$. This is useful for identifying locations present in both datasets (e.g., places visited by two users). Alternatively, *masking* zeroes counters where a binary filter indicates

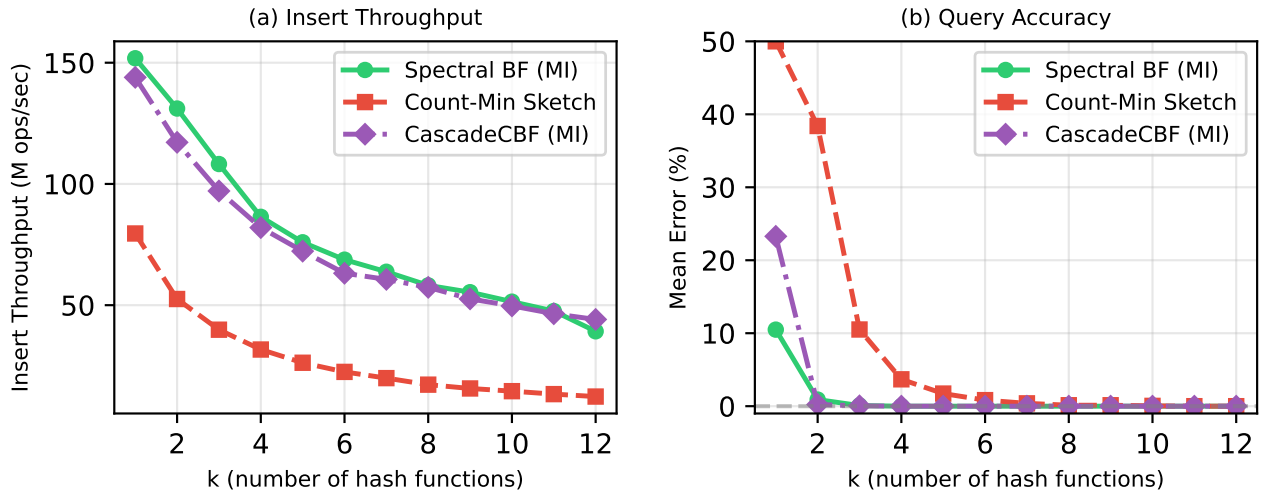

Figure 2. K-parameter sensitivity ($\alpha = 1.5$): (a) throughput decreases with k; (b) error decreases with k. CascadeCBF achieves negligible error (<0.01%) at $k \geq 4$.

absence, and *subtraction* computes set difference when one dataset is a superset.

Spatial Aggregation. Count-in-polygon queries sum GET results over all raster cells inside a polygon. Given polygon P rasterized to cells $\{c_1, \ldots, c_n\}$, the aggregate count is $\sum_{i=1}^{n} \text{GET}(c_i)$. This operation is embarrassingly parallel and can be accelerated via SIMD or GPU. The error accumulates additively: for n cells with mean overcount $\bar{\varepsilon}$, total error is $\mathcal{O}(n\bar{\varepsilon})$. In practice, large polygons (many cells) exhibit lower relative error since overcounting errors partially cancel.

Multi-Category Indexing. Extending coordinates with categorical attributes enables simultaneous tracking of multiple dimensions. A point $\langle x, y, c \rangle$ with category c hashes differently from $\langle x, y, c' \rangle$, providing complete category isolation–queries for category c return counts only for that category, not the total across all categories. This supports event classification (news categories), temporal binning (hourly counts), and multi-attribute queries without separate filters per category.

Spatial Movement of Datapoint. Due to the counting capabilities of the proposed CascadeCBFs, we can further move single points by deleting them in one location and adding them to another location. Additionally, a movement of all points in one location to a different location can be easily obtained with an element-wise addition of the underlying slots. A possible application of these spatial movements is a space-efficient tracking of spatial inventory numbers.

5 Evaluation

For the evaluation we compare CascadeCBF (with minimal increment) against Spectral Bloom Filters (SBF-MI) (Cohen and Matias, 2003) and Count-Min Sketch with conservative update (CMS) (Cormode and Muthukrishnan, 2005) first on a synthetic distribution before looking into the performance for real world applications. All experiments use $k = 8$ hash functions and 16-bit counters unless stated otherwise. We focus on insert and query accuracy for append-heavy geospatial streams (Section 6)

5.1 Synthetic Dataset

To evaluate the performance of the developed Cascade CBF in a defined environment we first evaluate them on Zipfian data (10K unique items, 100K insertions). Figure 2 shows that **CascadeCBF achieves negligible error (<0.01%) at** $k \geq 4$, matching SBF-MI, while CMS requires $k \geq 11$. This difference arises because CMS uses separate counter arrays per hash function (requiring more hashes to achieve the same coverage), whereas CascadeCBF and SBF share a single counter array across all k positions. The optimal $k^* \approx 0.693(m/n) \approx 4$ for our configuration explains why $k = 4$ suffices.

Standard mode CascadeCBF provides 1.5–2× higher throughput than SBF-MI due to simpler increment logic. With minimal increment mode enabled, throughput is comparable to SBF-MI, but accuracy improves significantly for low-k configurations where false positive collisions would otherwise accumulate. The difference stems from the minimal increment mode requiring a preliminary scan to find minimum counters before incrementing, whereas the standard mode increments all k positions unconditionally. Standard mode is omitted from Figure 2 as it trades accuracy for throughput.

Figure 3 shows error vs. memory across Zipfian distributions. Single-layer CascadeCBF with minimal increment (MI) matches the memory efficiency of SBF-MI and CMS: at $\alpha = 1.5$, all achieve <1% error with 32 KB.

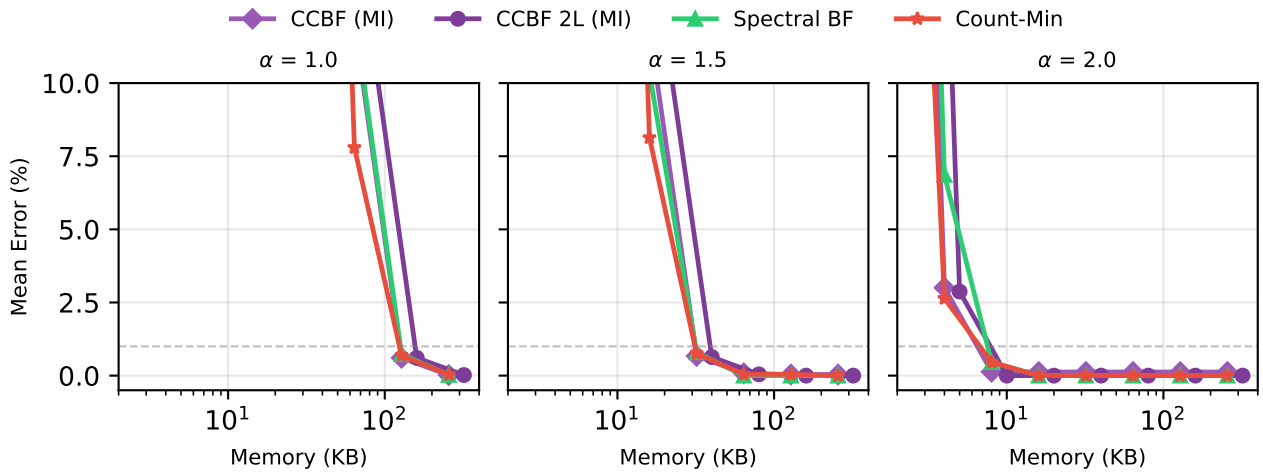

Figure 3. Memory efficiency across Zipfian skew levels. Each panel shows mean error vs. memory (log scale) for a different skew parameter α. Single-layer CascadeCBF (MI) matches SBF-MI and CMS accuracy. The two-layer variant (CCBF 2L) achieves near-zero error at high skew ($\alpha = 2.0$). All configurations use $k = 8$, 10K unique items, 100K insertions.

Two-layer CascadeCBF achieves near-zero error at higher skew ($\alpha = 2.0$) with just 10 KB by cascading overflow to a second layer.

This is particularly relevant for geospatial data, where population centers create extreme hotspots (e.g., $\alpha > 1.5$) that would overflow single-layer counters. NYC taxi data exhibits $\alpha \approx 2.0$: Times Square receives thousands of pickups daily while outer-borough streets see single digits. The multi-layer design adapts naturally: Times Square counters overflow to Layer 1 while suburban counters remain in compact Layer 0 storage.

5.2 Real-World Spatial Datasets

To align with the practical use of the proposed data structure in the geo domain, we evaluate on three real geospatial datasets: **GDELT** (Leetaru and Schrodt, 2013) (1.9M global news events with coordinates), **COVID-19** (Dong et al., 2020) (1.8M case locations sampled from 182M total) and **New York Taxi** (NYC Taxi and Limousine Commission, 2024) (987k taxi trips). All datasets exhibit spatial clustering: GDELT concentrates in major news regions (USA, Europe, Middle East), COVID-19 reflects population density with hotspots in India, Brazil, and the United States and New York Taxi has hotspots in mobility hubs like train stations and airports.

Table 1 shows insert times across all three datasets. CascadeCBF in standard mode achieves the fastest inserts, while MI mode is comparable to SBF-MI and CMS. Query times are similar across all implementations (∼0.27 ms). The 54× difference between GDELT and COVID-19 reflects higher point density causing more counter updates per cell.

In the following, we evaluate CascadeCBF across various real-world applications.

Table 1. Insert performance across datasets (seconds).

Implementation	GDELT (1.9M)	COVID-19 (1.8M)	NYC Taxi (987K)
CascadeCBF	0.13	6.99	–
CascadeCBF (MI)	0.18	9.22	0.07
Spectral BF (MI)	0.16	7.61	0.09
Count-Min Sketch	0.19	10.78	0.11

5.2.1 Multi-Category Event Classification

Real-world spatial data often includes categorical attributes (event types, time bins). We extend coordinates as ⟨lat, lon, category⟩ triplets, hashing the full vector to ensure category isolation. This approach requires no additional memory–categories share the same filter–while providing complete separation. We test on GDELT events classified into four types derived from Goldstein scale: Verbal Cooperation (46%), Material Cooperation (14%), Verbal Conflict (31%), Material Conflict (10%).

Table 3 shows CascadeCBF achieves the lowest error (0.02%) under constrained memory. At this resolution, maximum cell counts reach 33,051–exceeding the 32,768 threshold that triggers Layer 1 overflow. CascadeCBF's two 16-bit layers provide more counters than single 32-bit implementations at equal memory, while the overflow protocol handles extreme counts that would saturate single-layer filters.

[0] All implementations use 32-bit counters at 512 KB with $k = 8$ hash functions and 1-degree resolution (20K unique cells, max count 33K). Error is mean relative error across 10K queries per category.

Table 2. Time-series accuracy: mean error (%) at each snapshot.

Date	2020-03	2020-06	2020-12	2021-06	2021-12	2022-06
Cumulative	142	8K	92K	274K	563K	1.1M
Spectral BF (MI)	0.00	0.00	0.00	0.00	0.23	0.51
Count-Min Sketch	0.00	0.00	0.00	0.00	0.23	0.51
CascadeCBF (MI)	0.00	0.00	0.00	0.00	0.00	0.00

Table 3. Multi-category isolation on GDELT (1.9M events) at 512 KB, resolution=1.

Implementation	**Memory** (KB)	**Mean Err** (%)	**Max Err** (%)
CascadeCBF (MI)	511	0.02	100
Spectral BF (MI)	512	0.05	300
Count-Min Sketch	511	0.13	2100

5.2.2 Polygon Counting

Spatial aggregation queries sum counts within administrative boundaries. We insert 1.9M GDELT events into filters and query 1632 Natural Earth country polygons by summing counts across rasterized cells. Polygon rasterization converts boundaries to cell sets via scanline algorithms; each cell's count is queried and summed.

For large polygons, all implementations achieve low error: USA (626K events) shows 0.8% error with CascadeCBF, matching Spectral BF. Germany (45K events) shows 0.4% error, demonstrating consistent accuracy across scales. The mean error across all polygons is dominated by small/empty polygons where division by zero or small denominators amplifies noise.

For polygons with $>$1000 events, CascadeCBF achieves $<$2% error. CMS shows higher aggregation error (5–10%) due to systematic overcounting when summing many cells–each cell's false positive probability compounds additively. This makes CMS less suitable for large-scale spatial aggregation despite its strong theoretical guarantees for individual queries.

5.2.3 Time-Series Evolution

Power-law spatial data grows over time as events accumulate at hotspots. We track COVID-19 pandemic evolution across 6 snapshots (March 2020–June 2022), inserting 1.1M cumulative points. This tests the multi-layer overflow design under realistic growth patterns: early snapshots fit entirely in Layer 0, while later snapshots trigger overflow to Layer 1 as counts exceed 2^{15}.

Table 2 shows CascadeCBF maintains 0% error throughout all snapshots, while SBF-MI and CMS accumulate 0.51% error as counts grow to 1.1M. This validates the multi-layer overflow design: Layer 1 handles hotspot overflow while Layer 0 remains cache-efficient. The error growth in single-layer approaches is gradual–0% through mid-2021, then increasing as counters saturate. CascadeCBF's two-layer design provides headroom for continued growth beyond 1M points without accuracy degradation.

5.2.4 Urban Mobility Hotspots

Urban transportation data exhibits extreme spatial concentration–Manhattan receives $>$90% of NYC taxi pickups while outer boroughs remain sparse. This creates challenging conditions for probabilistic counting: hash collisions between dense Manhattan cells accumulate errors rapidly. We evaluate on 987K yellow taxi trips (NYC Taxi and Limousine Commission, 2024) from January 2015, rasterized to 4096$\times$4096 cells covering the NYC metropolitan area.

Table 4 shows all implementations achieve sub-1% mean error with comparable memory budgets (2 MB). CascadeCBF achieves 0.29% mean error with the fastest insert time (0.066s), while CMS shows slightly higher error (0.39%) due to its separate-array design accumulating more collisions. Single-cell queries at named hotspots–Times Square (23 pickups), Central Park South (60 pickups), Grand Central (7 pickups)–show exact counts for CascadeCBF and Spectral BF. Maximum errors (500–800%) occur at low-count cells where a single false positive collision dominates (e.g., true count 1, estimate 6); these affect $<$0.1% of queried cells and the 95th percentile error is $<$5%.

The key advantage of CascadeCBF in urban scenarios is throughput: 1.6$\times$ faster inserts than CMS (0.066s vs. 0.108s) due to shared counter arrays requiring fewer memory accesses per operation.

6 Discussion

When to Use CascadeCBF. CascadeCBF excels when (1) data follows power-law distributions with extreme hotspots, (2) counts span multiple orders of magnitude, or (3) spatial operations (union, aggregation) are needed. For uniform distributions or when provable error bounds are required, Count-Min Sketch may be preferable despite its higher error on skewed data.

Table 4. NYC Taxi: urban hotspot accuracy (987K pickups).

Implementation	Memory (KB)	Insert (s)	Mean Err (%)	Max Err (%)
Spectral BF (MI)	2048	0.085	0.28	500
CascadeCBF (MI)	2176	0.066	0.29	800
Count-Min Sketch	2041	0.108	0.39	800

Parameter Selection. The number of hash functions k controls the accuracy/throughput tradeoff. Our experiments show $k = 4$ suffices for typical spatial workloads. Layer sizing follows a simple heuristic: Layer 0 handles the common case (low counts), so its size m_0 should satisfy the Bloom filter optimality condition $m_0/n \approx 14$ for 1% false positives. Layer 1 can be significantly smaller–we use $m_1 = m_0/64$ since only $\sim$1% of elements overflow.

Memory vs. Accuracy. With 16-bit Layer 0 and 32-bit Layer 1 counters, CascadeCBF handles counts up to 2^{31} with 2.5 bytes per counter on average (Layer 0 dominates memory). Compared to fixed 32-bit CBF (4 bytes/counter), this saves 37.5% memory while providing exact counts for low-frequency items.

Limitations. CascadeCBF inherits Bloom filter limitations: (1) deletion requires standard mode–MI mode tracks only minimum counters, making safe decrement impossible; in standard mode, cascade decrement borrows from upper layers when Layer 0 underflows, (2) memory must be pre-allocated (unlike CMS which can stream), and (3) hash collisions cause systematic overcount for high-frequency items sharing hash positions. The multi-layer design mitigates but does not eliminate collision effects.

7 Conclusions

CascadeCBF provides efficient cardinality estimation for power-law spatial data through cascading overflow between layers. The overflow bit protocol enables compact storage while maintaining cache locality–most operations touch only Layer 0. With minimal increment mode, CascadeCBF achieves near-zero error ($<$0.01%) at $k \geq 4$, matching Spectral Bloom Filters while supporting counts up to 2^{30} with 1.3–1.6$\times$ higher throughput.

Evaluation on four real-world datasets spanning global news events, pandemic surveillance, and urban mobility demonstrates versatile spatial capabilities: (1) *multi-category indexing* on 1.9M GDELT events with perfect category isolation; (2) *polygon aggregation* with $<$2% error on country-level queries; (3) *time-series tracking* on COVID-19 data maintaining near-zero error as counts grow to 1.1M; and (4) *urban mobility* analysis on 987K NYC taxi trips with 1.6$\times$ faster throughput than Count-Min Sketch.

The multi-layer overflow design handles extreme spatial concentration characteristic of human mobility data. Where single-layer approaches must choose between overflow (small counters) and memory waste (large counters), CascadeCBF adapts per-element: most counters use compact Layer 0 storage while overflow cascades to Layer 1 only when needed.

Future work includes multiresolution re-randomization to restore hash uniformity across layers, deletion support via decrement operations, GPU-accelerated implementations for streaming spatial analytics, and integration with spatial databases for query optimization. We also plan to investigate learned hash functions that exploit spatial locality for improved cache performance on clustered data.

Use of AI Tools

Generative AI tools (Claude, Gemini) were utilized to assist with code development, debugging, and manuscript editing. All AI-generated content was reviewed, verified, and revised by the authors, who take full responsibility for the final work.

Acknowledgements

This work is partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 507196470.

References

Aguilar-Saborit, J., Trancoso, P., Muntes-Mulero, V., and Larriba-Pey, J. L.: Dynamic count filters, ACM SIGMOD Record, 35, 26–32, https://doi.org/10.1145/1121995.1122000, 2006.

Bloom, B. H.: Space/time trade-offs in hash coding with allowable errors, Communications of the ACM, 13, 422–426, https://doi.org/10.1145/362686.362692, 1970.

Bonomi, F., Mitzenmacher, M., Panigrahy, R., Singh, S., and Varghese, G.: An improved construction for counting bloom filters, in: Proceedings of the 14th Annual European Symposium on Algorithms (ESA), pp. 684–695, Springer, 2006.

Broder, A. and Mitzenmacher, M.: Network applications of bloom filters: A survey, Internet mathematics, 1, 485–509, 2004.

Cohen, S. and Matias, Y.: Spectral bloom filters, in: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, pp. 241–252, ACM, 2003.

Cormode, G. and Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications, Journal of Algorithms, 55, 58–75, 2005.

Dong, E., Du, H., and Gardner, L.: An interactive web-based dashboard to track COVID-19 in real time, The Lancet Infectious Diseases, 20, 533–534, 2020.

Fan, L., Cao, P., Almeida, J., and Broder, A. Z.: Summary cache: a scalable wide-area web cache sharing protocol, IEEE/ACM transactions on networking, 8, 281–293, 2000.

Ficara, D., Giordano, S., Procissi, G., and Vitucci, F.: MultiLayer Compressed Counting Bloom Filters, in: IEEE INFOCOM 2008 - The 27th Conference on Computer Communications, pp. 311–315, https://doi.org/10.1109/INFOCOM.2008.71, 2008.

Guo, D., Wu, J., Chen, H., Yuan, Y., and Luo, X.: The dynamic bloom filters, IEEE Transactions on Knowledge and Data Engineering, 22, 120–133, 2009.

Kirsch, A. and Mitzenmacher, M.: Less hashing, same performance: Building a better bloom filter, in: European Symposium on Algorithms, pp. 456–467, Springer, 2006.

Leetaru, K. and Schrodt, P. A.: GDELT: Global Data on Events, Location and Tone, 1979–2012, in: ISA Annual Convention, pp. 1–49, 2013.

NYC Taxi and Limousine Commission: TLC Trip Record Data, https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page, yellow taxi trip records 2009–2015 with pickup coordinates, 2024.

Werner, M.: Globimaps-A probabilistic data structure for in-memory processing of global raster datasets, in: Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 3–12, 2019.