**Comparative Evaluation Framework for Count-Min Sketch Variants in Streaming Data Environments**

by

Sergei Rogov

Final Year project submitted in partial fulfilment of the requirements for the Degree of

Bachelor of Science in Computer Science

Department of Computer Science

School of Sciences and Engineering

University of Nicosia

(June 2025)

***Comparative Evaluation Framework for Count-Min Sketch Variants in Streaming Data Environments***

***By***

***Sergei Rogov***

This Final Year Project has been accepted in partial fulfilment of the requirements for the Degree of

Bachelor of Science in Computer Science

Project Advisor:

\_\_\_\_\_\_\_\_Dr Demetris Trihinas / /\_\_

(Name) (Signature) (Date)

Examiner:

\_\_\_\_\_\_Professor Ioannis Katakis / /\_\_

(Name) (Signature) (Date)

Examiner:

\_\_\_\_\_\_\_\_\_Dr Andreas Savva / /\_\_

(Name) (Signature) (Date)

Department of Computer Science

School of Sciences and Engineering

University of Nicosia

**Abstract**

This project proposes a comparative evaluation framework for Count-Min Sketch and its variants, addressing the challenges of gathering statistics in streaming data environments. The framework is designed with a modular architecture, and can be used by researchers to compare the performance of different Count-Min Sketch variants or other summarization algorithms capable of answering frequency queries. The framework’s architecture includes multiple core modules such as input stream generation, algorithms implementation, performance metrics evaluation, and visualization component. For evaluation metrics, accuracy, memory usage, and average query time are monitored. Experiments are conducted on both real and synthetic datasets and real time visualisations are displayed on a dynamic dashboard during real-time stream data processing.

**Acknowledgements**

I would like to deeply thank my advisor, Dr. Demetris Trihinas, for his valuable feedback, technical guidance, and encouragement throughout the course of this Final Year Project. I am also extremely grateful to my parents for their unwavering moral and financial support during the entire period my studies, especially over the past year.

Table of Contents

[Chapter 1 Introduction 1](#_Toc200552181)

[1.1 Motivation 1](#_Toc200552182)

[1.2 Problem Statement 1](#_Toc200552183)

[1.3 Project Goals 2](#_Toc200552184)

[1.4 Scope and Limitations 2](#_Toc200552185)

[1.5 Structure of Report 2](#_Toc200552186)

[Chapter 2 Theoretical Background 5](#_Toc200552187)

[2.1 Characteristics of Streaming Data 5](#_Toc200552188)

[2.1.1 Volume 5](#_Toc200552189)

[2.1.2 Velocity 6](#_Toc200552190)

[2.1.3 Variety 6](#_Toc200552191)

[2.1.4 Veracity 6](#_Toc200552192)

[2.1.5 Value 6](#_Toc200552193)

[2.1.6 Variability 7](#_Toc200552194)

[2.1.7 Visualization 7](#_Toc200552195)

[2.1.8 Vulnerability 7](#_Toc200552196)

[2.1.9 Volatility 7](#_Toc200552197)

[2.2 Challenges in Real-Time Summarization 7](#_Toc200552198)

[2.3 Count-Min Sketch Approach 8](#_Toc200552199)

[2.3.1 2D array of counters 8](#_Toc200552200)

[2.3.2 Hash functions 8](#_Toc200552201)

[2.3.3 Add method 8](#_Toc200552202)

[2.3.4 Query method 9](#_Toc200552203)

[2.3.5 Count-Min Sketch Example 9](#_Toc200552204)

[2.3.6 Count-Min Sketch Characteristics 13](#_Toc200552205)

[Chapter 3 Framework Architecture 15](#_Toc200552206)

[3.1 Technology Stack 15](#_Toc200552207)

[3.1.1 Programming Language 15](#_Toc200552208)

[3.1.2 Core Libraries 15](#_Toc200552209)

[3.1.3 Web Dashboard 15](#_Toc200552210)

[3.1.4 Additional Libraries 16](#_Toc200552211)

[3.2 Modular Design Overview 16](#_Toc200552212)

[3.3 Stream Simulation Module 18](#_Toc200552213)

[3.3.1 Dataset Stream Simulator 19](#_Toc200552214)

[3.3.2 Random Stream Simulator 19](#_Toc200552215)

[3.4 Datasets 20](#_Toc200552216)

[3.4.1 FIFA World Cup 2018 Tweets 20](#_Toc200552217)

[3.4.2 uchoice-Kosarak 21](#_Toc200552218)

[3.5 Ground Truth Module 22](#_Toc200552219)

[3.5.1 Standard Truth 22](#_Toc200552220)

[3.5.2 Decaying Truth 22](#_Toc200552221)

[3.6 Algorithms Module 23](#_Toc200552222)

[3.6.1 Count-Min Sketch 24](#_Toc200552223)

[3.6.2 Conservative Count-Min Sketch 25](#_Toc200552224)

[3.6.3 Count-Mean-Min Sketch 27](#_Toc200552225)

[3.6.4 Count Sketch 28](#_Toc200552226)

[3.6.5 Hierarchical Count-Min Sketch 28](#_Toc200552227)

[3.6.6 Time Decay Count-Min Sketch 28](#_Toc200552228)

[3.7 Evaluation Module 28](#_Toc200552229)

[3.7.1 Accuracy 28](#_Toc200552230)

[3.7.2 Memory Usage 30](#_Toc200552231)

[3.7.3 Average Query Time 30](#_Toc200552232)

[3.8 Visualization Module 30](#_Toc200552233)

[3.9 Experiments Results Folder 30](#_Toc200552234)

[3.9.1 Folder Hierarchy 31](#_Toc200552235)

[3.10 Dashboard Module 32](#_Toc200552236)

[Chapter 4 Experiments and Results 33](#_Toc200552237)

[4.1 Optimal Parameters Selection 33](#_Toc200552238)

[4.2 Cross-Algorithm Comparisons 33](#_Toc200552239)

[4.3 Benchmark Summary Table 33](#_Toc200552240)

[4.4 Accuracy Trade-offs per Variant 33](#_Toc200552241)

[Chapter 5 Discussion 34](#_Toc200552242)

[5.1 Comparative Strengths 34](#_Toc200552243)

[5.2 Limitations of Variants 34](#_Toc200552244)

[5.3 Ideal Use Cases 34](#_Toc200552245)

[Chapter 6 Future Work 35](#_Toc200552246)

[6.1 Automatic Algorithm Selection 35](#_Toc200552247)

[6.2 Adaptive Resizing 35](#_Toc200552248)

[6.3 Semantic Mapping 35](#_Toc200552249)

[Chapter 7 Conclusion 37](#_Toc200552250)

[7.1 Summary of Contributions 37](#_Toc200552251)

[7.2 Major Findings 37](#_Toc200552252)

[References 38](#_Toc200552253)

[Appendices 40](#_Toc200552254)

**List of Illustrations**

Page

[Figure 1. Initialized Count-Min Sketch 10](#_Toc200552255)

[Figure 2. CMS after processing “foo” 10](#_Toc200552256)

[Figure 3. CMS after processing “bar” 11](#_Toc200552257)

[Figure 4. CMS after processing “foo” a second time 12](#_Toc200552258)

[Figure 5. Query("foo") 12](#_Toc200552259)

[Figure 6. Query("bar") 13](#_Toc200552260)

[Figure 7. System Architecture Diagram 18](#_Toc200552261)

[Figure 8. Experiments Folder Hierarchy 31](#_Toc200552262)

**List of Tables**

Page

[Table 1. Sample of pre-processed tweets from FIFA dataset 20](#_Toc200400554)

[Table 2. Sample from uchoice-Kosarak dataset 21](#_Toc200400555)

# Introduction

## Motivation

In the era of big data, where data is generated in rapidly growing volumes, with high velocity, and in various shapes and forms, there is an obvious need in efficient real-time analysis of data streams, as it faces big challenges in both computation and storage. Traditional methods of data storage and processing are inefficient because of the memory and processing speed limitations. However, approximate data stream summarization methods offer a more feasible solution, trading an adequate amount of accuracy for a significant improvement in memory utilization.

Popular example of such approximate data stream summarization algorithm is Count-Min Sketch (CMS) – a technique for efficiently answering frequency queries, meaning estimating how often an element appears in a stream, using limited memory. However, despite a wide use of CMS, there is no standardized framework for evaluation and comparing it with either its own variants or with other algorithms designed for frequency queries. The lack of such platform makes it difficult to determine the most suitable algorithm for a specific case.

## Problem Statement

There is a need in a standardized evaluation framework for detailed comparison and research of different variants of Count-Min Sketch summarization algorithm. Such platform will enable researchers and developers to better understand the strengths and weaknesses of different variants of approximate summarization techniques in different scenarios, as well as compare them in a clear visual manner, considering parameters like accuracy, memory usage, and average time taken to answer a query.

## Project Goals

The project aim is to develop a comparative evaluation tool with modular architecture for benchmarking multiple CMS variants. The framework should include:

* Generation of a data stream, both synthetically and from real sources
* Implementation of Count-Min Sketch algorithm and its variants
* Evaluation module, monitoring algorithm performance, tracking key metrics
* Real-time visualisation dashboard for easier interpretation and comparison

## Scope and Limitations

The framework covers a selected set of Count-Min Sketch algorithm variations and does not explore other summarization techniques with similar functionality. The work focuses on numeric and text data streams where elements are only added and never removed. For any element in the stream, the algorithm can only perform two operations:

* Update its internal to reflect the inclusion of the item
* Query the item to provide its estimated frequency in the stream

## Structure of Report

The report consists of seven core chapters, each targeting a specific aspect of a Count-Min Sketch comparative framework.

* Chapter 1 – Introduction

This chapter provides a high-level overview of project’s context and motivation, stating the specific problem this work aims to address. It explains the pressing demand in efficient data stream processing algorithms as well as the need in a framework for their detailed comparison for the purpose of both research and decision-making regarding the selection of a better suiting algorithm variation for a certain scenario.

* Chapter 2 – Background

This section introduces the concept of streaming data and challenges associated with processing it. Besides, it provides an overview of the Count-Min Sketch core principles, as well as the explanation on how exactly this family of algorithms operate on a data structure level. Related work is also covered in this chapter.

* Chapter 3 – Framework Architecture

This chapter is dedicated to the explanation of the modular architecture of the framework and the technology stack used during the project implementation, including the programming languages and libraries used. The purpose and the structure of each module is discussed, as well as the communication between components.

* Chapter 4 – Experiments and Results

This section presents a series of experimental studies carried out under different conditions, conducted using the implemented framework. Various parameters values are tested and optimal ones are determined, and the behaviour of different Count-Min Sketch variations is demonstrated on multiple data streams of different modalities, both for streams generated from real and synthetic datasets.

* Chapter 5 – Discussion

The chapter is dedicated to the analysis and the evaluation of the results from the previous chapter. Strengths and limitations of different Count-Min Sketch variants are shown, and the ideal use cases for algorithm variations are identified.

* Chapter 6 – Conclusion

Major findings and key takeaways from the work carried out are discussed in this chapter. The project summary is presented, highlighting the main milestones and results.

* Chapter 7 – Future Work

The closing chapter proposes the future work that can potentially extend the functionality of the framework implemented, expanding its scope by adding more advanced techniques and capabilities to the platform.

# Theoretical Background

## Characteristics of Streaming Data

In the modern day and age, the phenomenon of Big Data has fundamentally changed data-driven system across various domains, such as finance, healthcare, Internet of Things, E-commerce, and others. Big Data offers significant advantages by enabling specialists to extract meaningful insights from data and to leverage the gained knowledge for the purposes such as workflow optimisation, revenue growth, or predicting human behaviour based on the common patterns mined from vast datasets. However, alongside these benefits, Big Data introduces challenges, particularly in more difficult data processing and data storage tasks. Streaming data, as a dynamic and time-sensitive subset of Big Data, adds even more complexities because of its continuous and high-speed nature.

Streaming data is often characterized by the “V” model, each dimension of which starts with the letter “V” [1].

### Volume

The problem with the overall amount of data being produced. Streaming systems must handle extremely large volumes of data generated from wide range of sources, such as sensors, social media platforms, mobile devices. Without the optimized algorithms, specifically tailored high-throughput environments, it would not be feasible to deal with such data streams. In many real-world scenarios, storing raw streaming data is either impractical or cost-prohibitive, making the use approximation techniques necessary to reduce memory utilization with a sacrifice of accuracy.

### Velocity

The challenge that refers to the of speed at which streaming data arrives and should be processed. In many cases, such as traffic monitoring or fraud detection, data should be handled instantaneously, in a single pass, as any latency may lead to the loss of critical information and inability to access data again. Efficient and low latency algorithms are crucial when dealing with velocity problem.

### Variety

Data could appear in a structured (e.g., tables), semi-structured (e.g., JSON, XML), and unstructured formats (e.g., text, video, audio), creating a challenge of inconsistency – handling such heterogeneous data types requires a high level of flexibility in processing logic.

### Veracity

The problem of data quality, which refers to the data reliability and accuracy. Not all the sources can be trusted, because the information coming from the source that lacks credibility could lead to faulty conclusions and decisions. Besides, data streams could contain noise and inconsistencies, requiring data stream processors to include filtering, anomaly detection and validation techniques to mitigate erroneous outcomes from streaming data handling.

### Value

Some data is less useful than the other. Distinguishing between low-value data and high-value data in real time is crucial, because processing resources are usually limited. In order to effectively make use of data, streaming systems should identify which sections of data carry the most insights. Useful patterns should only be found and extracted from most informative parts of data streams. For example, irrelevant or outdated data should be ignored.

### Variability

Data streams may fluctuate because of seasonal effects, user behaviour or other external events. Adaptive streaming models are required for these scenarios for better accuracy and memory utilisation.

### Visualization

The problem of presenting the big data to the target audience in a clear and understandable way. This involves highlighting key trends and showcasing overall picture while omitting noise and irrelevant details that do not covey any insights.

### Vulnerability

Data streams are vulnerable to cyberattacks. It is essential not only to handle the data itself, but also to protect it in transit, applying security measures.

### Volatility

Streaming data has an important property – once missed, it cannot be restored. That is why it is crucial for stream handling systems to perform real-time low latency data processing.

## Challenges in Real-Time Summarization

Since the handling of streaming data enforces memory limitations and latency constrains, fast and lightweight approximations are required. Exact methods are often too slow and memory-intensive, which means they are not applicable in real-time data streaming environments. Therefore, approximation techniques like sketching or sampling, which trade off accuracy for speed and efficiency, must be used. Designing such algorithms introduces new challenges as it requires wise balancing between memory error bounds, memory consumption and computational overhead.

## Count-Min Sketch Approach

Basic Count-Min Sketch (CMS) summarization algorithm has a clever design, allowing it to answer frequency queries in an approximate manner, with a fixed memory utilization budget. It works with numeric and text data streams, or any data streams consisting of hashable items.

Basic CMS algorithm consists of the following attributes and methods.

### 2D array of counters

The main data structure of the algorithm, which is used to store the counters associated with items from the data stream. The matrix has dimensions *depth* × *width*, meaningit consists of *depth* rows and *width* columns. This 2D array is updated or queried by the algorithm methods to maintain frequency estimates.

### Hash functions

The CMS algorithm instance has *depth* different hash functions, each associated with each of the *depth* rows in a 2D matrix of counters. These hash functions are used to hash data items and map them to each of *depth* rows of length *width*, either to update CMS with the item being processed or to query the item.

### Add method

In order to update the CMS instance with the new item, the method is used.

Each hash function *hj* maps this item to a column index in the row , effectively allocating this object to one of the *width* buckets in that row, where . Once the insert positions are determined, the corresponding counters in the 2D array are incremented:

More generally, if the item being processed is of quantity :

By the nature of the algorithm and due to the memory constraints, the width of the 2D counter array is limited, making hash collisions unavoidable.

### Query method

In order to query the item in the CMS instance, the method is used.

To answer a frequency query, algorithm hashes an item using each of the *depth* hash functions, mapping it to *depth* different positions in the 2D array of counters. Then, it retrieves the counter values from these positions and returns the minimum value among them as an estimated frequency :

.

Due to the hash collisions, overestimations can occur because multiple objects might be hashed to the same cell, causing their frequency count values to be combined. However, returning the minimum across all hashed positions helps to mitigate this effect.

### Count-Min Sketch Example

To illustrate how Count-Min Sketch works, consider a small sketch with 5 buckets per row and 3 hash functions ().

#### Step 1. Initialization

Before the stream processing, the data structure is initialized as a 2D array of zeros. See Figure 1.

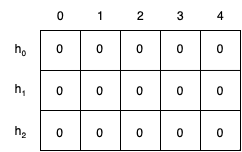


Figure 1. Initialized Count-Min Sketch

#### Step 2. Processing “foo”

The item arrives. Count-Min Sketch calls and updates the 2D array of counters.

Suppose, “ is hashed to the following positions:

Counters at positions , , and are incremented, like shown in Figure 2.

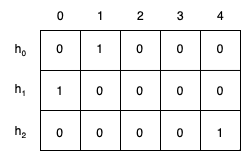
****

Figure 2. CMS after processing “foo”

#### Step 3. Processing “bar”

Item “ arrives. It hashes to:

Count-Min Sketch counters at positions , , and are incremented. See Figure 3.

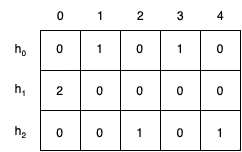


Figure 3. CMS after processing “bar”

#### Step 4. Processing “foo” again

When arrives again, it is hashed to the same positions as in Step 2. Counters , , and are incremented, as demonstrated on Figure 4.

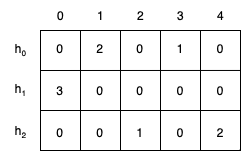


Figure 4. CMS after processing “foo” a second time

#### Estimating Frequencies

To determine how many times an element has appeared in the stream, the method is used. It hashes an item times to determine cells associated with this element. Then it looks up the counters in these positions and returns the minimum value among them.

See Figure 5.

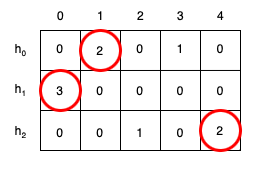


Figure 5. Query("foo")

See Figure 6.

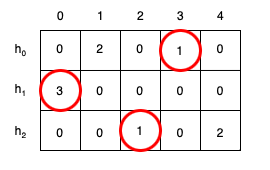


Figure 6. Query("bar")

Even though row experienced a hash collision ( and shared the same cell ), Count-Min Sketch still provided correct estimated frequencies. This is because of two other collision-free rows – their presence prevented overestimations.

### Count-Min Sketch Characteristics

Classic Count-Min Sketch has two accuracy parameters , which are used to set the dimensions of the 2D matrix. and

The answer to point query is given by .

The estimate has the following guarantees:

, and, with probability at least , ,

where:

* is the ground true frequency of item ,
* is the estimated frequency,
* is the sum of all frequencies in a stream,
* is an accuracy parameter, which controls the maximum additive error,
* is an accuracy parameter controlling failure probability.

This means the algorithm never underestimates. Count-Min Sketch guarantees that estimates are always equal to or greater than the true frequencies, but never exceed them beyond the overestimation bound specified by parameter . [2]

In different CMS variants, the implementations of the components discussed above may vary, depending on the particular strategy employed. Definitions and error guarantees of other Count-Min Sketch variants will follow in Chapter 3, Section 3.6.

# Framework Architecture

## Technology Stack

### Programming Language

The project is developed using Python programming language. Python offers great flexibility, rich ecosystem of libraries, and is widely considered to be one of the most suitable programming languages for the data processing and visualisation tasks.

### Core Libraries

Multiple Python libraries were utilized during the development of the framework:

* NumPy – A fundamental library for scientific computing in Python, for performing efficient numeric operations.
* Matplotlib – A comprehensive library for creating static, animated, and interactive visualizations in Python.
* hashlib – A module that implements a common interface to various hash algorithms.
* unittest – Python’s built-in unit testing framework.

### Web Dashboard

As the framework includes a real-time visualisation module, the web-based dashboard libraries are used:

* Plotly – An open-source graphing library for Python for creating interactive graphs.
* Dash – A Python low-code framework built on top of Flask, Plotly and React for rapidly building interactive web-based applications.

### Additional Libraries

Throughout the project, the following standard Python libraries were additionally utilised for a wide range of system-level and utility-level tasks:

argparse, json, os, datetime, copy, csv, heapq, random, time, abc, subprocess

## Modular Design Overview

The framework’s architecture is organized into distinct modular components following the Single Responsibility Principle for the purpose of scalability and maintainability. Each module performs a specific role and encapsulates its related functionality. The interaction between these components is illustrated in Figure 5.

The workflow is initiated by a Dashboard Module, which serves as the system’s frontend. Here, the user selects experiment parameters and initiates the simulation by pressing the “Run Experiment” button. Simulation module then acts as an orchestrator, integrating other components and performing the following tasks:

* Requests Stream Simulation Module to generate a data stream, either numeric or textual, depending on user’s selection
* Accesses Algorithms Module to retrieve the implementation of the selected algorithm
* Requests Ground Truth Module to provide a data structure instance for tracking true element frequencies
* Interacts with the Evaluation Module to assess algorithm performance using three evaluation metrics:
  + Accuracy compared to ground truth
  + Memory usage
  + Average query time

Stream Simulation Module reads from a dataset folder, which contains various dataset files, and converts this static data into a simulated data stream.

During the data stream processing, the Simulation module periodically writes evaluation results in format to Experiment Results folder. It also periodically requests Visualization Module to generate static plots, which are saved in the same folder.

Finally, the Dashboard Module periodically reads the results from the Experiment Results folder to update the dynamic charts displayed in a frontend.

The experiment can be terminated by the user via “Stop Experiment” button.

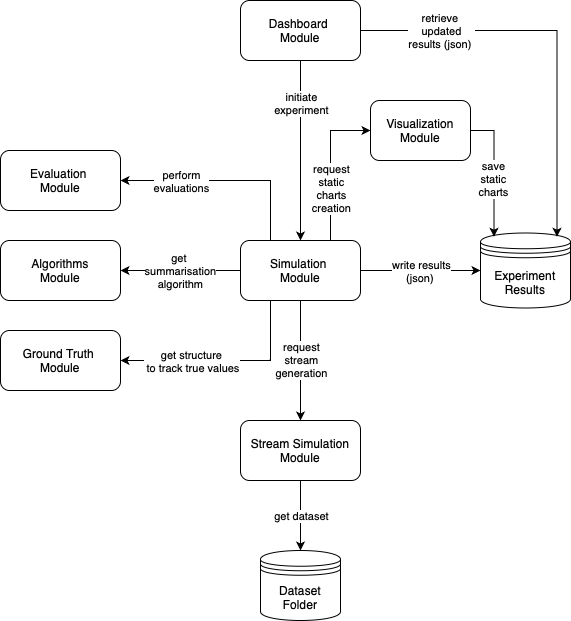


Figure 7. System Architecture Diagram

## Stream Simulation Module

The Stream Simulation Module is responsible for generating real-time data streams. Instead of relying on live inputs, the system simulates streaming behaviour by yielding items from a predefined source with controlled timing. This design allows algorithms to be tested in reproducible environments. The simulated streams are generated from two types of sources: static datasets or synthetic data.

The module defines an abstract interface , which specifies the common behaviour for all stream simulators. It is implemented by two concrete classes, and .

### Dataset Stream Simulator

generates a real-time data stream by reading from a static dataset and yielding elements sequentially with a fixed time delay. supports the following file formats:

* CSV

For the Comma Separated Values files, it reads the specified field from each row, and yields each encountered word in that field, one at a time with a small delay.

* TXT

For text datasets, simulator reads the file line by line, splitting each line into tokens (words), which are then yielded sequentially at a constant rate.

### Random Stream Simulator

produces a synthetic data stream with skewed element frequency distribution, modelled using a Zipfian distribution. The aim of such generation is to mimic the real-world data, which is often distributed unevenly, with some items having a much higher frequency than the others.

This simulator creates a stream of a fixed size with a configurable skewness parameter .  
Generation of data stream with skewed data distribution. Like in the dataset-based simulator, emits items sequentially with a small delay to emulate the real-time flow.

## Datasets

This section describes the datasets used in the project to simulate data streams. Two real-world datasets and one synthetically generated dataset are employed.

### FIFA World Cup 2018 Tweets

This dataset consists of 530,000 tweets in English, related to the 2018 Football World Cup, which were collected over the span of time from the Round of 16 to the final stage. Tweets in the dataset have been pre-processed and stored in the column of the CSV file. Peculiarities like website names, hashtags, user mentions, special characters, trailing/heading/multiple spaces were removed from tweets. Additionally, contractions such as *n’t*, *’ll*, *’ve, ‘re* have been expanded to their full English forms. Although the dataset contains 16 attributes in total, only the column containing pre-processed text is the subject of interest for this work for the task of counting the occurrences of individual words. The sample of the dataset is demonstrated in Table 1. [3]

Table 1. Sample of pre-processed tweets from FIFA dataset

|  |  |
| --- | --- |
| ID | Tweet |
| 1013597060640145408 | Only two goalkeepers have saved three penalties in penalty shoot out Ricardo vs |
| 1013597056219295744 | scores the winning penalty to send into the quarter finals where they will face Russia |
| 1013597047482544130 | Tonight we have big game |
| 1013597044198391808 | We get stronger Turn the music up now We got that power power |
| 1013597039999926272 | Only two goalkeepers have saved three penalties in penalty shoot out Ricardo vs |
| 1013597039995867143 | We re looking strong going into the knockout stage We caught up with ahead of |
| 1013597039978995712 | am happy for winning Especially since you know we colluded and all Russia eliminates Spain after pen... |
| 1013597038951436288 | When you see me When we feel the same feeling Power power |
| 1013597038188154880 | Kasper Schmeichel takes the final award of the day |
| 1013597037118525440 | After Years Global Puma Ambassador LG Mobile Ambassador CocaCola WorldCup Kookmin Bank UNICEF |

### uchoice-Kosarak

The uchoice-Kosarak text dataset contains 505,217 subset selections over a universal set of 2,605 integer items. Each line in the dataset is a space-separated list of integers representing a set of links on a Hungarian news portal visited by a user in a given browsing session.

The dataset exists in two versions:

* uchoice-Kosarak – unfiltered version.
* uchoice-Kosarak-5-25 – a filtered version, in which each subset selection contains at most 5 items and all items appear in at least 25 selections.

Both versions were used in the experiments. Each integer element is treated individually in this work, as the main task is to count item appearances. The sample of the dataset is shown in Table 2. [4]

Table 2. Sample from uchoice-Kosarak dataset

|  |
| --- |
| Subset Selection |
| 1 2 3 |
| 1 |
| 4 5 6 7 |
| 1 8 |
| 9 10 |
| 11 6 12 13 14 15 16 |
| 1 3 7 |
| 17 18 |
| 11 6 19 20 21 22 23 24 |
| 1 25 3 |

## Ground Truth Module

The Ground Truth Module provides the data structure for tracking the true number of item appearances in the stream, serving as a reference for evaluating how accurately summarization algorithms perform. Within the experimental setup, the Ground Truth Module achieves full accuracy in tracking item frequencies, since it is not limited by the resource constraints that affect approximate algorithms. The module defines the interface as a template for two types of data structures holding the element frequencies: standard frequency counter and a decaying window-based one.

### Standard Truth

Standard Truth data structure is represented by a Python dictionary with items as keys and the number of appearances of these items as values. This data structure stores element counts from the beginning of a particular stream processing.

### Decaying Truth

Decaying Truth, similarly to Standard Truth, is a Python dictionary with items as keys and the number of appearances of these items as values. However, this data structure is used for sliding window scenarios, when only the recent elements are important and old items are ignored. uses a and only remembers last items, automatically discarding the old ones when the window gets full.

## Algorithms Module

The Algorithms Module is the core part of the framework, responsible for storing and managing implementations of different Count-Min Sketch (CMS) variants. It provides a common interface through the abstract base class , ensuring consistency across CMS variations. The interface enforces all CMS variants to provide the implementations of the following methods:

A method for updating the CMS data structure with the element encountered in the stream. An item may be counted with the optional weight , which shows the quantity of the item to be added. By default, , meaning adding only one instance of the element. While different CMS variations may implement it differently depending on a strategy, every variant must support this method.

A method for estimating the item frequency in the stream, returning the approximate count.

Additional method for clearing all the stored counts, resetting the CMS data structure to its initial state.

A method for determining the load factor of the CMS data structure, defined as the ratio of the buckets with non-zero counters in the most occupied to the total number of buckets in that row (). This method helps to monitor the utilization and density of CMS.

The following sections cover the implementations of multiple CMS variants.

### Count-Min Sketch

#### Algorithm

The base Count-Min Sketch concept was fully illustrated in Chapter 2, Section 2.3. The algorithm:

* Initializes a 2D array of zeros ()

This matrix is employed to store counters of items from the stream.

* Uses method to update with the element being processed

Item is hashed by independent hash functions to cell positions, one per row. The corresponding buckets in the 2D array are incremented. If the item arrives with the quantity of (by default ), counters in corresponding buckets are increased by :

* Uses method to return an estimated frequency of an element in the stream

This procedure is performed by hashing an item to positions, identifying the values of counters in those positions and returning the minimum value among them:

.

#### Benefits

* Probabilistic guarantees

The key benefit lies in its probabilistic guarantees, controlled by accuracy parameters and . For a detailed mathematical exposition, refer to Chapter 2, Section 2.3.6.

* Space efficiency

The algorithm uses a sublinear space compared to the total number of distinct elements in the stream, making it suitable for handling large datasets in situations where storing exact counts is memory-prohibitive.

* Time efficiency

Both and methods are extremely fast, operating with time complexity, making them suitable for high throughput streaming environments.

#### Limitations

* Overestimation bias

With the increasing number of items processed, 2D array can experience a significant amount of hash collisions, leading to highly inflated estimated counts, especially for rare elements.

* Dependence on Hash Functions

Not carefully chosen or not independent hash functions could lead to increased rate of hash collisions, worsening the algorithm’s estimation accuracy.

* Lack of Adaptability

Once initialized, the 2D matrix has a fixed dimensions, which cannot be easily resized without re-initialization.

### Conservative Count-Min Sketch

#### Algorithm

Conservative Count-Min Sketch is an extension of a base Count-Min Sketch algorithm, modelled to decrease the overestimation bias. Conservative CMS employs a distinguished strategy for the method, while the other functionality has the same implementation as in standard CMS.

* Initialization of the 2D array

Identical to the base CMS - 2D array of zeros () to store the counters.

* method

Unlike CMS, conservative CMS performs 3 distinct steps to update the 2D matrix of counters:

* + Step 1: Hash the item to positions

Using independent hash functions, hash the item to buckets of a 2D array of counters, one per row – same as in standard CMS.

* + Step 2: Determine the current estimated frequency

Determine the current estimated frequency of the item by querying the sketch (find the minimum value among all counters associated with the item being processed).

* + Step 3: Conservatively update relevant counters

Each counter corresponding to the item being processed is set to the maximum value between its current value and the estimated frequency increased by the incoming quantity :

This operation prevents some counters falling below the global minimum of the particular item frequency among counters values. Effectively, only those counters less than will be incremented.

* method

Query method employs the same implementation as in standard CMS. The procedure is performed by hashing an item to positions, identifying the values of counters in those positions and returning the minimum value among them.

#### Benefits

* Reduced overestimation effect

Conservative CMS significantly reduced the overestimation bias because of its conservative update strategy as it limits the impact of collisions for individual items. This generally improves algorithm’s accuracy.

#### Limitations

* Increased update complexity

The conservative update strategy requires additional query for finding the current minimum value before modifying counters. This makes more computationally intensive than standard CMS.

### Count-Mean-Min Sketch

#### Algorithm

#### Benefits

#### Limitations

### Count Sketch

#### Algorithm

#### Benefits

#### Limitations

### Hierarchical Count-Min Sketch

#### Algorithm

#### Benefits

#### Limitations

### Time Decay Count-Min Sketch

#### Algorithm

#### Benefits

#### Limitations

## Evaluation Module

The Evaluation Module is used to monitor the performance of the implemented Count-Min Sketch variants based on three key metrics: accuracy, memory usage and average query time. These metrics are essential for understanding the trade-offs of different CMS variations. Evaluation Module contains three sub-modules.

### Accuracy

The accuracy metric measures the error between the estimated frequency of an item and its true frequency in the data stream. To compute this, the module compares the predicted frequency of each item with its ground truth count. Multiple factors associated with accuracy metric are calculated and reported:

* Overestimation/underestimation/exact match percentage

The fraction of all estimated counts that has a value greater than/less then/equal to the actual item frequency.

* Average Error

The mean of absolute values of errors. A metric that demonstrates the value of the error on average.

* Average Error Percentage

The average error percentage is calculated in the following way – each item’s absolute error value is divided it by ground truth frequency to determine a relative error, which is then converted it to a percentage and averaged over all items. Essentially, this metric shows an average relative error.

* Maximum Error Percentage

The value of the highest relative error is demonstrated by this metric.

* Overestimation/underestimation/combined percentiles

For a better understanding of the distribution of estimated errors, three sets of percentiles are computed: for overestimation, underestimation, and for their combination.

Overestimations are considered directly, while underestimation are converted to absolute values. Then, both overestimation and underestimation are merged to create a combined list of absolute errors.

For each group, 50th (median), 90th. 95th and 100th (maximum) percentiles are calculated.

* Top 20 over- and underestimations

Two lists with the 20 items with the largest overestimations and with the 20 items with the largest underestimations. This statistic helps to identify the most inaccurately estimated elements in the stream.

### Memory Usage

This sub-module measures the total memory consumed by a Count-Min Sketch instance. The memory usage equals to the total memory size of the 2D array of counters. In the case of non-adaptive Count-Min Sketch implementations, where depth and width dimensions of the 2D matrix of counters are fixed, the memory utilisation stays constant throughout the stream processing.

### Average Query Time

This sub-module measures the time taken to answer a frequency query using a Count-Min Sketch instance on average. It randomly samples a number of items from the ground truth, measures the time required to query these items sequentially, and provides an average value of time taken to perform a single query operation.

## Visualization Module

Visualization Module is responsible for providing a visual analysis of CMS algorithm performance by generating static line plots based on the metrics collected by the evaluation module. After generation, each graph is saved to the Experiments Results Folder in the format.

## Experiments Results Folder

The Experiments Results Folder serves as a central repository for storing experiment outputs. It is organized into a tree hierarchy.

### Folder Hierarchy

#### Dataset Level

A top-level directory, named after a dataset used in experiment.

#### Algorithm Level

Inside each dataset folder, subfolders correspond to the Count-Min Sketch variant used in experiment.

#### Parameters Level

Under algorithm folder, each subfolder corresponds to the particular parameter combination used in experiment, with the naming convention , where the number after letter represents width and the number after corresponds to the depth of a Count-Min Sketch.

#### Timestamp Level

This tree level separates experiments by the time and date they were executed. The name format is *2025-05-01\_16-17-46*.

Each timestamped folder stores both:

* Numerical results of the experiment in format
* Visualization plots representing the experiment in format

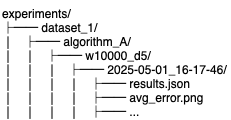


Figure 8. Experiments Folder Hierarchy

## Dashboard Module

The Dashboard Module provides a dynamic interface for real-time monitoring of key evaluation metrics during the data stream processing. It is designed to compare the performance of two selected CMS algorithms variations by displaying their evolving metrics side by side through interactive plots, providing clear and intuitive visual comparison.

The dashboard consists of a control panel and graphs displayed side by side.

### Control Panel

### Dynamic Graphs

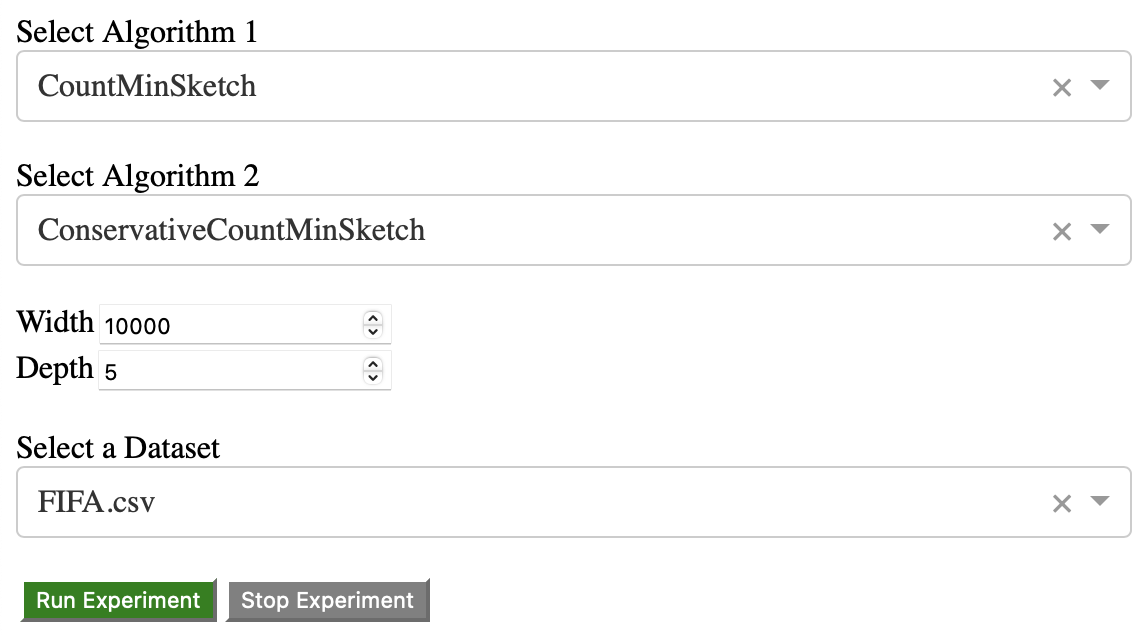


Figure 9. Control Panel

## Simulation Module

# Experiments and Results

## Optimal Parameters Selection

Empirical tuning of CMS width and depth parameters.

#### Width

#### Depth

## Cross-Algorithm Comparisons

Side-by-side evaluation under identical data conditions.

## Benchmark Summary Table

Tabular overview of key metrics across all variants.

## Accuracy Trade-offs per Variant

Analysis of performance variations per algorithm.

# Discussion

## Comparative Strengths

Each algorithm’s niche strengths outlined based on empirical results.

## Limitations of Variants

Discussion of scenarios where certain CMS types underperform.

## Ideal Use Cases

Recommendations based on data type, skewness, and resource constraints.

# Future Work

In this section, several promising directions for potential extensions of this project are presented.

## Automatic Algorithm Selection

The idea is to develop heuristics or classifiers that can automatically choose the most suitable Count-Min Sketch variant depending on the type and nature of the data or on the properties of the stream. For instance, the tool could suggest a user to employ a specific CMS variant depending on the factors such as whether the stream is numeric or natural language, and whether the item distributions is uniform or highly skewed.

## Adaptive Resizing

This project extension entails dynamically modifying Count-Min Sketch’s width and depth dimensions in response to variations in the properties of the data stream. For instance, if the stream experiences a sudden rise in the number of distinct elements occurrences or item frequency skew increases, CMS’s width should be increased to accommodate more items and reduce hash collisions. Conversely, if the stream becomes smaller in size and less variable, it might be appropriate to reduce CMS’s dimensions to save memory while preserving accuracy. The goal is to adapt the CMS parameters over time in order to maintain accuracy and efficient memory utilization as the data stream evolves.

## Semantic Mapping

This extension investigates the replacement of traditional hash-based indexing used in Count-Min Sketch with semantic item clustering. The algorithm could use semantic similarity to group objects together based on their word embeddings, such as Word2Vec or BERT, rather than considering each word in the input stream as a distinct token. This clustering would increase computational complexity compared to simple hashing, which might affect the performance in terms of average query time.

Semantic mapping is especially beneficial for natural language data streams, because in this environment, comprehending clusters-level counts of linked words is often more important than knowing the frequencies of individual words. This approach can potentially improve the accuracy of queries such as estimating the combined frequency of similar words, synonyms, or related terms.

# Conclusion

## Summary of Contributions

Outlines the framework’s capabilities and its role in CMS benchmarking.

## Major Findings

Highlights key differences observed among the variants.

# References

|  |  |
| --- | --- |
| [1] | R. Jati, “Exploring The Potential of Hyperlocal Media: Benefits, Challenges, and The Future Directions,” *JOURNAL OF HUMANITIES SOCIAL SCIENCES AND BUSINESS (JHSSB),* vol. 2, no. 2, pp. 475-480, 2023. |
| [2] | M. Nowak, “Flutter vs. React Native in 2025 — Detailed Analysis,” Nomtek, January 2024. [Online]. Available: https://www.nomtek.com/blog/flutter-vs-react-native. [Accessed April 2024]. |
| [3] | J. Richman, “Firebase vs. MongoDB: Major Differences,” Estuary, 03 May 2023. [Online]. Available: https://estuary.dev/blog/firebase-vs-mongodb/. [Accessed May 2025]. |
| [4] | V. Lo, “The Better Firebase? Introduction to Clerk.js,” Articles by Victoria Lo, January 2025. [Online]. Available: https://lo-victoria.com/introduction-to-clerkjs. [Accessed May 2025]. |
| [5] | Stack Overflow, “2024 Stack Overflow Developer Survey,” 2024. [Online]. Available: https://survey.stackoverflow.co/2024/. |
| [6] | J. Nielsen, “Response Times: The 3 Important Limits,” January 1993. [Online]. Available: https://www.nngroup.com/articles/response-times-3-important-limits/. [Accessed April 2025]. |
| [7] | OWASP, “Authentication · OWASP Cheat Sheet Series,” Owasp.org, 2017. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Authentication\_Cheat\_Sheet.html. |
| [8] | European Union, “General Data Protection Regulation (GDPR),” 2016. [Online]. Available: https://gdpr.eu/. |

<http://dimacs.rutgers.edu/~graham/pubs/papers/timedecaypodc.pdf>

<https://www.cs.princeton.edu/courses/archive/spr04/cos598B/bib/CharikarCF.pdf>

<https://aclanthology.org/D12-1100.pdf>

[1] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," Mobile Networks and Applications, vol. 19, no. 2, pp. 171–209, 2014.

[2] 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, Apr. 2005.

[3] <https://www.kaggle.com/datasets/rgupta09/world-cup-2018-tweets>

[4] A Discrete Choice Model for Subset Selection.  
Austin R. Benson, Ravi Kumar, and Andrew Tomkins.  
In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining (WSDM)*, 2018. [[bibtex](https://www.cs.cornell.edu/~arb/data/bibs/higher-order-choice-wsdm-2018-bib.txt)]

https://www.cs.cornell.edu/~arb/data/uchoice-Kosarak/

[n] M. Kleppmann, Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol, CA, USA: O’Reilly Media, 2017.

# Appendices