**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](#_Toc200315080)

[1.1 Motivation 1](#_Toc200315081)

[1.2 Problem Statement 1](#_Toc200315082)

[1.3 Project Goals 2](#_Toc200315083)

[1.4 Scope and Limitations 2](#_Toc200315084)

[1.5 Structure of Report 2](#_Toc200315085)

[Chapter 2 Theoretical Background 5](#_Toc200315086)

[2.1 Characteristics of Streaming Data 5](#_Toc200315087)

[2.1.1 Volume 5](#_Toc200315088)

[2.1.2 Velocity 6](#_Toc200315089)

[2.1.3 Variety 6](#_Toc200315090)

[2.1.4 Veracity 6](#_Toc200315091)

[2.1.5 Value 6](#_Toc200315092)

[2.1.6 Variability 7](#_Toc200315093)

[2.1.7 Visualization 7](#_Toc200315094)

[2.1.8 Vulnerability 7](#_Toc200315095)

[2.1.9 Volatility 7](#_Toc200315096)

[2.2 Challenges in Real-Time Summarization 7](#_Toc200315097)

[2.3 Count-Min Sketch Approach 8](#_Toc200315098)

[2.3.1 2D array of counters 8](#_Toc200315099)

[2.3.2 Hash functions 8](#_Toc200315100)

[2.3.3 Add method 8](#_Toc200315101)

[2.3.4 Query method 9](#_Toc200315102)

[2.3.5 Count-Min Sketch Example 9](#_Toc200315103)

[2.3.6 Count-Min Sketch Characteristics 12](#_Toc200315104)

[Chapter 3 Framework Architecture 14](#_Toc200315105)

[3.1 Technology Stack 14](#_Toc200315106)

[3.1.1 Programming Language 14](#_Toc200315107)

[3.1.2 Core Libraries 14](#_Toc200315108)

[3.1.3 Web Dashboard 14](#_Toc200315109)

[3.1.4 Additional Libraries 15](#_Toc200315110)

[3.2 Modular Design Overview 15](#_Toc200315111)

[3.3 Stream Generation Module 15](#_Toc200315112)

[3.3.1 Real Dataset Source 15](#_Toc200315113)

[3.3.2 Synthetic Dataset Source 15](#_Toc200315114)

[3.4 Algorithms Module 15](#_Toc200315115)

[3.4.1 Count-Min Sketch 15](#_Toc200315116)

[3.4.2 Conservative Count-Min Sketch 16](#_Toc200315117)

[3.4.3 Count-Mean-Min Sketch 16](#_Toc200315118)

[3.4.4 Count Sketch 16](#_Toc200315119)

[3.4.5 Hierarchical Count-Min Sketch 16](#_Toc200315120)

[3.4.6 Time Decay Count-Min Sketch 16](#_Toc200315121)

[3.5 Evaluation Module 17](#_Toc200315122)

[3.6 Visualization Module 17](#_Toc200315123)

[Chapter 4 Experiments and Results 18](#_Toc200315124)

[4.1 Optimal Parameters Selection 18](#_Toc200315125)

[4.2 Cross-Algorithm Comparisons 18](#_Toc200315126)

[4.3 Benchmark Summary Table 18](#_Toc200315127)

[4.4 Accuracy Trade-offs per Variant 18](#_Toc200315128)

[Chapter 5 Discussion 19](#_Toc200315129)

[5.1 Comparative Strengths 19](#_Toc200315130)

[5.2 Limitations of Variants 19](#_Toc200315131)

[5.3 Ideal Use Cases 19](#_Toc200315132)

[Chapter 6 Conclusion 20](#_Toc200315133)

[6.1 Summary of Contributions 20](#_Toc200315134)

[6.2 Major Findings 20](#_Toc200315135)

[Chapter 7 Future Work 21](#_Toc200315136)

[7.1 Automatic Algorithm Selection 21](#_Toc200315137)

[7.2 Adaptive Resizing 21](#_Toc200315138)

[7.3 Semantic Mapping 21](#_Toc200315139)

[Appendices 22](#_Toc200315140)

[References 23](#_Toc200315141)

**List of Illustrations**

Page

[Figure 1. Initialized Count-Min Sketch 10](#_Toc200306811)

[Figure 2. CMS after processing “foo” 10](#_Toc200306812)

[Figure 3. CMS after processing “bar” 11](#_Toc200306813)

[Figure 4. CMS after processing “foo” a second time 11](#_Toc200306814)

**List of Tables**

# 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.

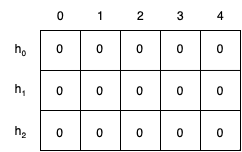


Figure 1. Initialized Count-Min Sketch

#### 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.

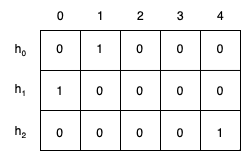
****

Figure 2. CMS after processing “foo”

#### Step 3. Processing “bar”

Item “ arrives. It hashes to:

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

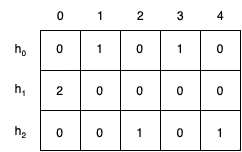


Figure 3. CMS after processing “bar”

#### Step 4. Processing “foo” again

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

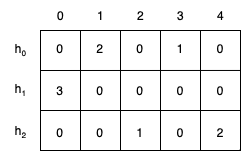


Figure 4. CMS after processing “foo” a second time

#### 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.

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 initiated the simulation by pressing the “Run Experiment” button. Simulation module then acts as an orchestrator, performing the following tasks by integrating other components:

* 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 JSON 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 reads the JSON results from the Experiment Results folder to update the dynamic charts displayed in a frontend.

To terminate the experiment, user can press the “Stop Experiment” button.

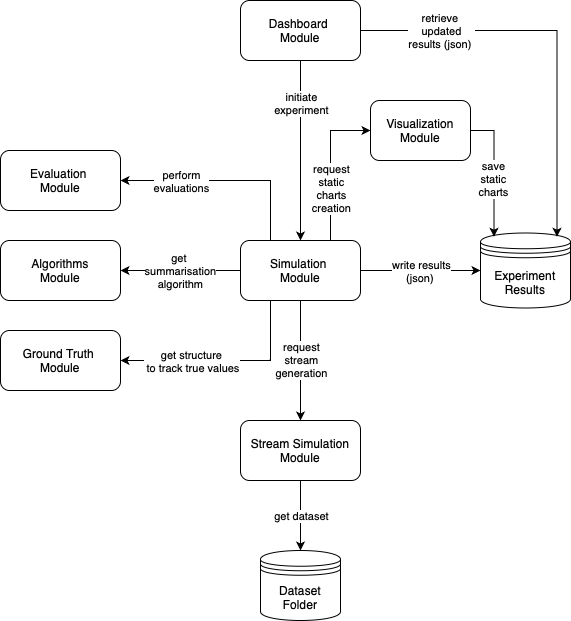


Figure 5. System Architecture Diagram

## Stream Simulation Module

The Stream Simulation Module is responsible for generating real-time data streams. Instead of relying on live data, the system simulates streaming behaviour in one of two ways: either from static datasets, or through synthetic generation. This is useful for testing algorithms in reproducible and controlled environments.

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 time delay.

### Random Stream Simulator

, generating a synthetic data stream with skewed element frequency distribution.  
Generation of data stream with skewed data distribution.

## Datasets

### FIFA Tweets

### Kosrak

### New

## Ground Truth Module

## Algorithms Module

The module defines an abstract interface

### Count-Min Sketch

#### Algorithm

#### Benefits

#### Limitations

### Conservative Count-Min Sketch

#### Algorithm

#### Benefits

#### Limitations

### 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

Contains three components

#### Accuracy

Comparison of predicted item frequencies with ground truth.

#### Memory Usage

Total memory utilized to store a Count-Min Sketch instance.

#### Average Query Time

Time taken to answer a frequency query on average.

## Visualization Module

#### Graphs Generation

Multiple charts showing algorithm behaviour.

#### Dynamic Dashboard

Real-time dashboard displaying all metrics of two algorithms chosen.

## Experiments Results Folder

## Dashboard 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.

# Conclusion

## Summary of Contributions

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

## Major Findings

Highlights key differences observed among the variants.

# Future Work

## Automatic Algorithm Selection

Heuristics for picking the best CMS variant per data type or per dataset type.

## Adaptive Resizing

Dynamic adjustment of sketch dimensions.

## Semantic Mapping

Semantic item clustering instead of hashing.

# Appendices

# 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.

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