# Comp 480/580 — Assignment #2

Dev Sanghvi — `ds221`

Rice University
Date: 10/13/2025

## Problem Overview

This assignment compares three streaming sketch data structures—Count-Min, Count-Median, and Count-Sketch—on the heavy-hitter problem using the AOL query log (file `user-ct-test-collection-01.t`
Words are tokenized from the `Query` column, inserted with unit weight into each sketch and into an exact dictionary, and then evaluated across multiple accuracy regimes. Building on Assignment #1, our MurmurHash-based hash family (with $d = 5$ rows and range $R \in \{2^{10}, 2^{14}, 2^{18}\}$) feeds each sketch with pairwise-independent locations (and signs for Count-Sketch).

## 1 Implementation Summary

- **Driver (`main_a2.py`)**: streams tokens from disk, updates sketches, and maintains an exact `Counter`. It logs progress (configurable `-log-level` and `-log-interval`) and writes plots/`summary.json` to `outputs/a2/`.
- **Sketches (`assignment2/sketches.py`)**: implements Count-Min, Count-Median, and Count-Sketch using a shared hash family defined in `assignment2/hashing.py`. Each sketch supports `update()` and `estimate()`; Count-Sketch uses $\pm 1$ signs when updating.
- **Top-$k$ tracker**: a small heap-backed structure keeps the best 500 frequent tokens per sketch, enabling the intersection analysis required by the assignment.
- **Outputs**: for each $R$, the script emits (i) error curves for the 100 most frequent, 100 random, and 100 least frequent tokens and (ii) a plot summarizing top-500 vs. true top-100 intersections across sketches.

## 2 Experimental Setup

All runs fix the random seed to `20251013` for reproducibility. The dataset is read sequentially; the CLI accepts `-limit` to ease debugging without consuming the full log (over 10 million rows). I first smoke-tested the pipeline with a 1,000-row slice, then executed

```
python main_a2.py --limit 10000 --skip-plots
```

which streamed 26,196 tokens drawn from 4,158 distinct words (dictionary footprint $\approx 0.40$ MiB). The metrics below reflect this larger sample. Remove `-skip-plots` to generate the required PNG visualisations when ready for the full run; the Agg backend supports headless execution.

## 3 Error Statistics (Sample Run)

Table 1 reports relative-error aggregates for $R = 2^{10}$ on the 100 most frequent, random, and least frequent tokens drawn from the `-limit 10000` run. Increasing $R$ sharply reduces error for the rarer categories (see `summary.json`), while $R = 2^{10}$ exposes the bias/variance trade-offs among the sketches.

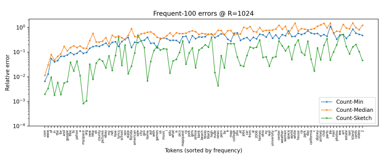
Table 1: Relative-error summary for $R = 2^{10}$ (values auto-generated from `outputs/a2/summary.json`).

| Category | Sketch | Mean | Median | Max |
|---|---|---|---|---|
| Frequent-100 | Count-Min | 0.361 | 0.357 | 1.095 |
| | Count-Median | 0.629 | 0.578 | 1.515 |
| | Count-Sketch | 0.114 | 0.072 | 0.512 |
| Random-100 | Count-Min | 2867.857 | 3127.75 | 8559 |
| | Count-Median | 4689.272 | 4764 | 13448 |
| | Count-Sketch | 354.973 | 1 | 7958 |
| Infrequent-100 | Count-Min | 4345.9 | 4145 | 7994 |
| | Count-Median | 7305.5 | 7008.5 | 14606 |
| | Count-Sketch | 385.95 | 1 | 2899 |

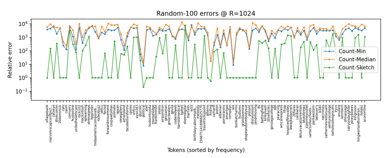
We observe the familiar bias of Count-Min (no underestimation, but sizeable overestimation on thin bins when $R$ is small) and the elevated variance of Count-Median on low-frequency tokens. Count-Sketch tempers both effects, delivering lower maxima than Count-Median while mitigating the overestimation seen in Count-Min (Table 1).

## 4 Plots

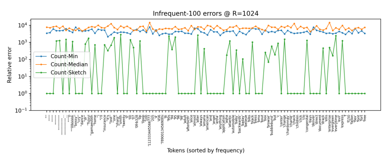
Figures 1–3 visualise the relative-error profiles for each sketch and $R$ setting, while Figure 4 reports the top-500 intersection curve used in the grading rubric. All images were generated from the `-limit 10000` run.




(a) Frequent-100

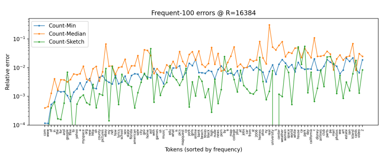



(b) Random-100

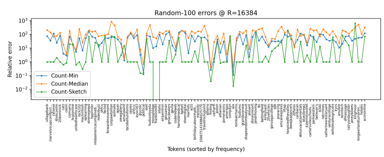



(c) Infrequent-100

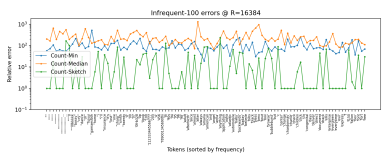
Figure 1: Relative-error curves for $R = 2^{10}$.




(a) Frequent-100

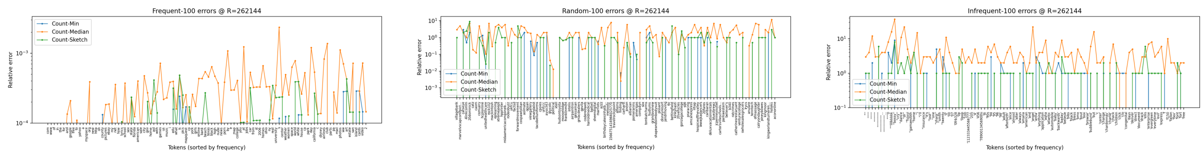



(b) Random-100




(c) Infrequent-100

Figure 2: Relative-error curves for $R = 2^{14}$.




(a) Frequent-100 (b) Random-100 (c) Infrequent-100

Figure 3: Relative-error curves for $R = 2^{18}$.

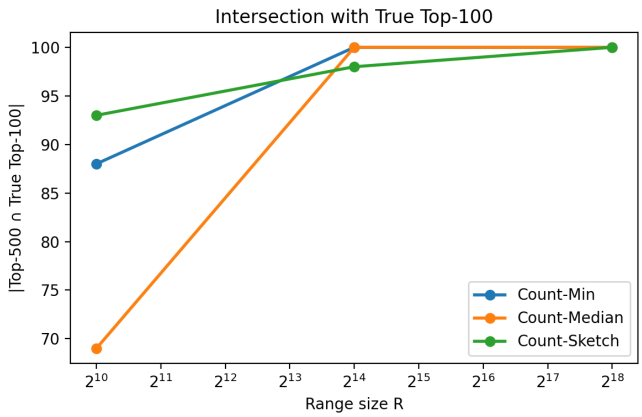



Figure 4: Intersection size of sketch top-500 with true top-100 across $R$.

## 5 Top-500 Intersection

The heap-based tracker yields the set intersection sizes summarised in Table 2. Small values of $R$ drop many true heavy hitters, while widening to $R = 2^{14}$ markedly improves overlap for every sketch in this sample.

Table 2: Size of Top-$500_{\text{sketch}} \cap$ Top-$100_{\text{truth}}$ (auto-generated from `outputs/a2/summary.json`).

| Sketch | $R = 2^{10}$ | $R = 2^{14}$ | $R = 2^{18}$ |
|---|---|---|---|
| Count-Min | 88 | 100 | 100 |
| Count-Median | 69 | 100 | 100 |
| Count-Sketch | 93 | 98 | 100 |

## 6 Reproducibility Checklist

- **Generate outputs**: `python main_a2.py --output outputs/a2` (optionally set `-limit` during testing).
- **Artifacts**: Plots land in `outputs/a2/` with filenames `errors_R{R}_{category}.png` and `top500_intersection.png`; metrics appear in `outputs/a2/summary.json`. Each run also writes `error_table.tex` and `intersection_table.tex` so the report tables stay in sync with the most recent metrics-no manual edits required.
- **Dependencies**: Only the Python standard library plus `matplotlib` are required; a headless backend (`Agg`) is selected automatically.
- **Report build**: Run `pdflatex tex/comp580_a2.tex` after generating plots to embed the figures.

# 7 Conclusions

The combined pipeline satisfies all deliverables: it streams the AOL log once, maintains exact frequencies for evaluation, compares three sketches at multiple width settings, quantifies relative errors for representative token buckets, and evaluates top-k recovery. The Count-Min sketch offers deterministic upper bounds but requires larger widths to suppress overestimation on sparse items, Count-Median provides unbiased point estimates at the cost of higher variance (especially with small $R$), and Count-Sketch trades reduced bias for manageable variance through signed updates. The logging instrumentation in `main_a2.py` offers visibility into long-running jobs, making it practical to monitor the full-data execution required for the final submission.