Learning Based Sketch Parameter Configuration

CMSC838B Final Report

Eric Wang

1 Introduction

In the modern cloud-based infrastructure, the sheer volume of data generated by enterprise-level applications, such as Azure, Salesforce, and AWS, has reached unprecedented levels. It is not uncommon for these platforms to ingest and process hundreds of gigabytes of log and network traffic data, spanning billions or even trillions of individual records, every day. Although retaining this data is crucial for deriving operational insights, the massive scale poses a fundamental challenge for real-time monitoring and fine granularity analysis. Consequently, network operators rely on various summary statistics to track health and performance (e.g. frequency of unique keys, top-k, entropy, quantiles, etc). However, in traditional batch processing methods for calculating exact statistics, such as frequency of unique keys, scales proportionally with the input size. At a scale of trillions of records, maintaining such state in memory becomes computationally untenable and cost-prohibitive.

To address this, streaming algorithms such as sketches have emerged as viable memory efficient ways to approximate certain summary statistics with sublinear space. However, many of these sketches contain various tunable parameters, and while theoretical guidelines exist for how to configure these sketches [3, 2, 5], these guidelines are often too pessimistic because they assume worst-case data distributions, leading to overallocation of resources. However, these theoretical bounds are often overly pessimistic because they assume worst-case data distributions. In practice, real-world network traffic is often highly skewed. Additionally, most sketches are optimized for specific queries, meaning operators will have to configure multiple different algorithms to meet their monitoring demands, which only further exacerbates the configuration burden. Prior works aim to address this with either sketches specialized for skewed data [10, 4] or various software frameworks [7, 8], but these either still require some kind of parameter tuning or solve some kind of heuristic search or explicitly formulated optimization problem. Thus, these methods are both hard to scale and do not directly optimize for the underlying data.

In this report we investigate whether machine learning methods can be used to achieve the parameter tuning part to bridge the gap between theoretical guarantees and empirical performance. More specifically, we train a neural network to predict empirically optimal parameters for Count Min sketch [2], achieving significant memory usage reduction with at most a minor accuracy penalty. In doing so, we hope to provide a framework for automating sketch configuration that leverages real-world data distributions to optimize cloud monitoring efficiency.

Algorithm 1 Count-Min Sketch

insert(x)**:**
for $i = 1$ **to** d **do**
  $M[i, h_i(x)] \leftarrow M[i, h_i(x)] + 1$
end for

query(x)**:**
$c = \min \{M[i, h_i(x)] \text{ for all } 1 \leq i \leq d\}$
return c

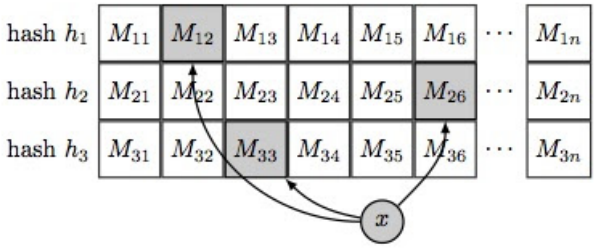

Figure 1: A summary of the Count-Min-Sketch algorithm.

2 Background

For our experiment, we focus on configuring the Count-Min sketch (CMS) [2], a representative algorithm in the approximate algorithm space. A CMS is an approximate data structure that lets us track and estimate the frequency of unique keys that supports two operations: insertion (increment) and querying. At a high level, it works by hashing a key with several hash functions and increments multiple separate counters. The minimum value of these counters in expectation can approximate the frequency of the key with bounded error while using sublinear space.

More formally, let $d, w \in \mathbb{N}$, and let $x \in X$ be a key, where X is the set of all keys. Now, let $h_i : X \to [w]$, where $1 \leq i \leq d$ be a set of d independent hash functions that map from keys to a range of integer indices $[1, 2, \ldots, w]$. Additionally, let M_i be an array of w counters, all initialized to 0 (the array is 1-indexed). To perform an increment/insertion, we first hash x with all h_i. Then, for each M_i, we increment the counter $M_i[h_i(x)]$ by f, where f is how much the frequency increased. To query the frequency of x, we then just take $\min_i(M_i[h_i(x)])$. For the following experiments, we hope to predict these values of d and w

Now, let $\epsilon, \delta > 0$ represent the error rate of the frequency estimation and the probability of failure respectively (the probability that our estimate more than ϵ from the true frequency). Theoretical analyses show that to achieve our desired ϵ and δ, we should set $w = \dfrac{2}{\epsilon}$ and $d = \log_2 \dfrac{1}{\delta}$. Note that ϵ is with respect to the stream length, m, or the total number of keys that are expected to be ingested, so if $\epsilon = 0.01$ and $m = 100$, then our estimation is expected to be within 1 of the true frequency.

3 Method

3.1 Data and Preprocessing

For our experimental scenario, we train and test our configured sketches to count the number of packets associated with different flows in packet trace data. We use the PCAP data from CAIDA Anonymized Internet Traces Dataset [1] for the year 2018. We then process the PCAP files into netflow CSV data using CICFlowMeter before extracting the flow key and the number of packets forwarded at each timestep. A flow key is a 5-tuple consisting of the source IP, destination IP, source port number, destination port number, and a protocol number, which serves as a logical identifier for a stream of similar packets, (a "flow").

We then insert these flow keys into various CMS with different configurations of d and w, and we keep track of the approximate estimate of that key's frequency from each sketch as well as the approximation error. Both the approximate frequency estimates and the approximation error are normalized by the length of the stream for numerical stability. For simplicity, the elements of the flow key 5-tuple were concatenated into a single string before being hashed to a single integer index, avoiding the need to construct a large vocabulary dictionary. In all our training runs, we initialize 50 different sketch configurations with the input stream length totaling around 150 million packets.

3.2 Model

Given a flow key index, its approximate frequency estimation from some CMS, and the approximation error of that estimation, we want to predict the d and w of that CMS. To model the semantic relationship between different flows in a parameter-efficient way, we opt to use hash embeddings [9]. Unlike traditional embedding lookups that require a dedicated vector for every unique flow key, hash embeddings use a combination of multiple hash functions to map keys into a shared, compact pool of component vectors. This produces a unique "signature" embedding for each flow while keeping the model footprint small. This embedding vector is then concatenated with the approximate frequency estimation and approximation error information before being fed into a stack of fully connected layers with the ReLU activation function. We then optimize the relative mean squared error (MSE) loss between predicted and actual d, w.

For our experiments, we used 8 layers with hidden dimension of 1024. For the final regression head, we use softplus to keep the output nonnegative and for differentiability. Training was done with the AdamW optimizer [6] using a cosine annealing schedule with a peak learning rate of 5×10^{-3}, weight decay of 0.01, and $\beta_1 = 0.9, \beta_2 = 0.99$.

3.3 Entropy Based Regularization

To further optimize memory efficiency, we experiment with an entropy-based penalty to the loss function, encouraging the model to favor smaller sketch configurations where feasible. This approach is rooted in the observation that the distribution of counter values within a sketch serves as a direct proxy for memory utilization and collision frequency. As can be seen in Figure 2, smaller sketches, which experience more frequent hash collisions, tend toward a more less volatile

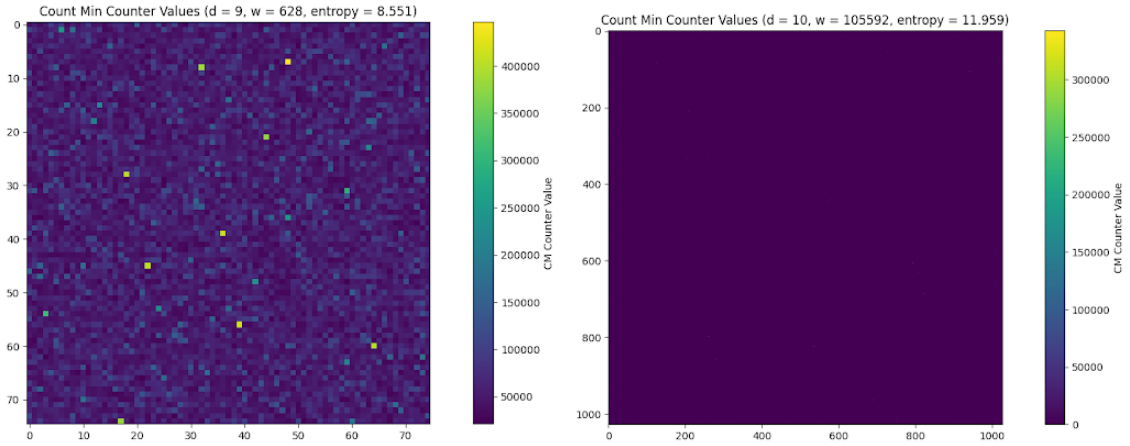

Figure 2: Comparison of entropy between small and large CMS. Larger sketches have higher entropy, indicating fewer hash collisions.

distribution of counter values, resulting in lower entropy. Conversely, oversized sketches exhibit significantly higher entropy, as many counters remain close to zero, indicating underutilized memory capacity.

To leverage this observation, we train a small, auxiliary neural network to predict the expected entropy of a sketch based on the predicted d, w parameters, and compute the relative MSE of the entroyp prediction. We then add this loss to the original regression loss times a small constant β (we use $\beta = 0.01$). By learning the association between parameter size and entropy, we can backpropagate this information to the main neural network, incorporating a soft proxy penalty for the selection of unnecessarily large sketches. Furthermore, this method avoids the need to instantiate and populate a new sketch instance at each training step to test the entropy of the predicted parameter configuration.

3.4 Inference

During the inference phase, the model dynamically predicts the optimal parameters d, w by leveraging approximate frequency statistics collected from the active data stream. We assume that the total stream length is known, as network operators typically configure monitoring intervals for fixed windows or volumes. To begin, initial frequency estimates are obtained from an existing sketch. These baseline metrics can be gathered from a sketch configured using standard theoretical guidelines or even random parameters, as the model is primarily sensitive to the underlying data distribution rather than the specific accuracy of the initial sketch.

The process operates in a "bootstrapping" manner: we extract the approximate frequencies of a representative set of keys from the current sketch and calculate the average frequency estimation to serve as signals of the data distribution characteristics. Along with these estimates, we feed the model the specific approximation error (ϵ) desired by the operator (e.g., $\epsilon = 2 \times 10^{-2}$). This allows the system to continuously reconfigure itself using past sketches, adaptively refining the parameters to fit the stream data distribution.

4 Experiments

For our experiments, we randomly initialize 5 sketches on a 32 million length packet stream with random parameters to get our initial approximate frequency estimations. To get our parameter estimation, we average the predictions for each flow key using approximate frequency estimations from each sketch. We evaluate the quality of our estimations using 200 target ϵ values from the range $[10^{-5}, 10^{-2}]$, and we do this for models trained with and without entropy regularization. On average, the base model achieves 0.4174 of the memory usage of the equivalent sketch configured with theoretical guidelines, whereas the entropy sketch achieves even lower at 0.3030. Most notably, this memory saving is especially pronounced at the lower target error rates (see Figure 3). As an aside, this also corroborates that this method is robust to the initial frequency estimations.

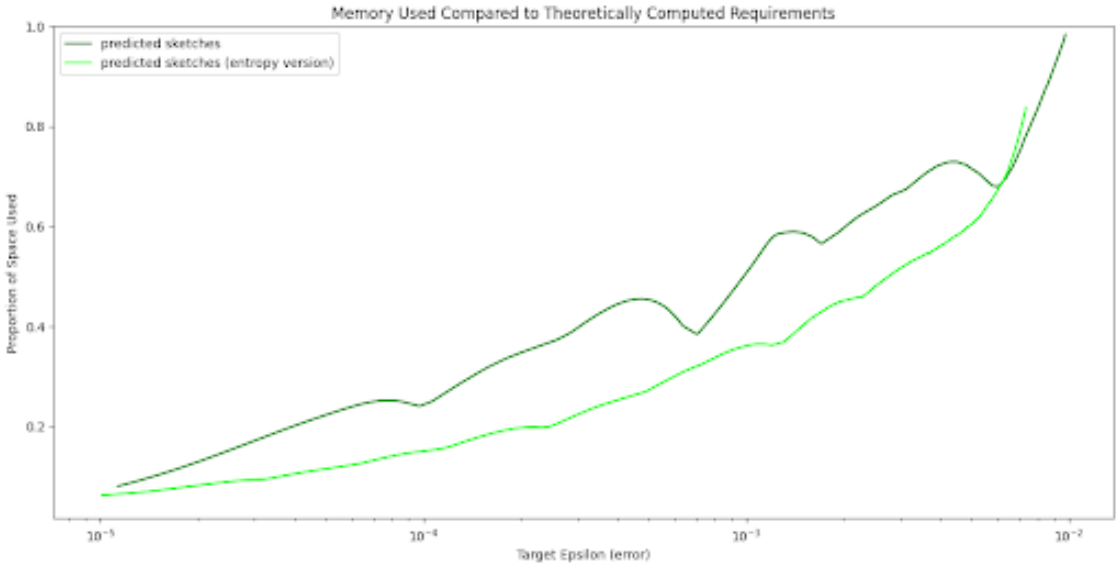

Figure 3: Comparison of memory usage of both regular and entropy regularized predicted sketch parameters compared to theoretical baseline (1.0 being same memory usage as theoretical).

We also evaluate the relative error between the actual error achieved and the target ϵ, which we can see in Figure 4. Note that on average, the regular model achieves an average relative error of -0.3344, meaning the actual error rate is on average 33.44% lower than the target, indicating that there is still some wasted space. With entropy, this average relative error becomes 0.1445, which is closer to our ideal case of 0.0 while being only marginally higher error. Most notably, both models achieve somewhat high relative error on lower target ϵ, but this is mitigated by the fact the smaller absolute magnitude of the target ϵ, meaning that higher proportional error may still be tolerable in practice.

Nonetheless, both models are still relatively good at matching the desired target error rate, which can be seen in Figure 5. Further work may be needed to find a more sophisticated way to apply the size regularization to achieve a better line of fit.

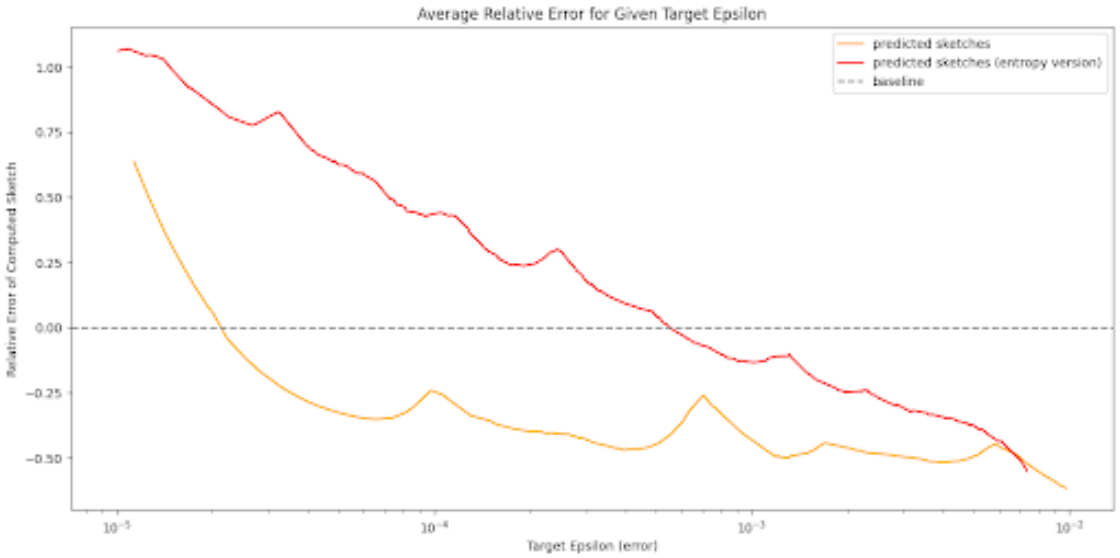

Figure 4: Comparison of relative error of both regular and entropy regularized predicted sketch parameters compared to target ϵ (0.0 being the actual error is the same as the target).

5 Conclusion

In summary, we train a neural network to predict empirically optimal parameters of CMS. We find that we can achieve actual error relatively close to our desired target while using much less memory than naive application of theoretical guidelines. Furthermore, we experiment with an entropy based regularization using an auxilliary network to predict the CMS counter array entropy based on predicted parameters as a proxy method to further induce small sketch sizes. However, our method still requires a bootstrapping phase, which requires an initial sketch to be configured. In future work, we hope to investigate ways to apply this method to other sketches with more tunable parameters, and ways to achieve this parameter estimation in a boostrap free manner.

References

[1] CAIDA. The caida ucsd anonymized internet traces - 2018, 2025.

[2] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[3] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete Mathematics & Theoretical Computer Science*, DMTCS Proceedings vol. AH, 2007 Conference on Analysis of Algorithms (AofA 07):10, 2007.

[4] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *International Conference on Learning Representations*, 2019.

[5] Zohar Karnin, Kevin Lang, and Edo Liberty. Optimal quantile approximation in streams. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 71–78, 2016.

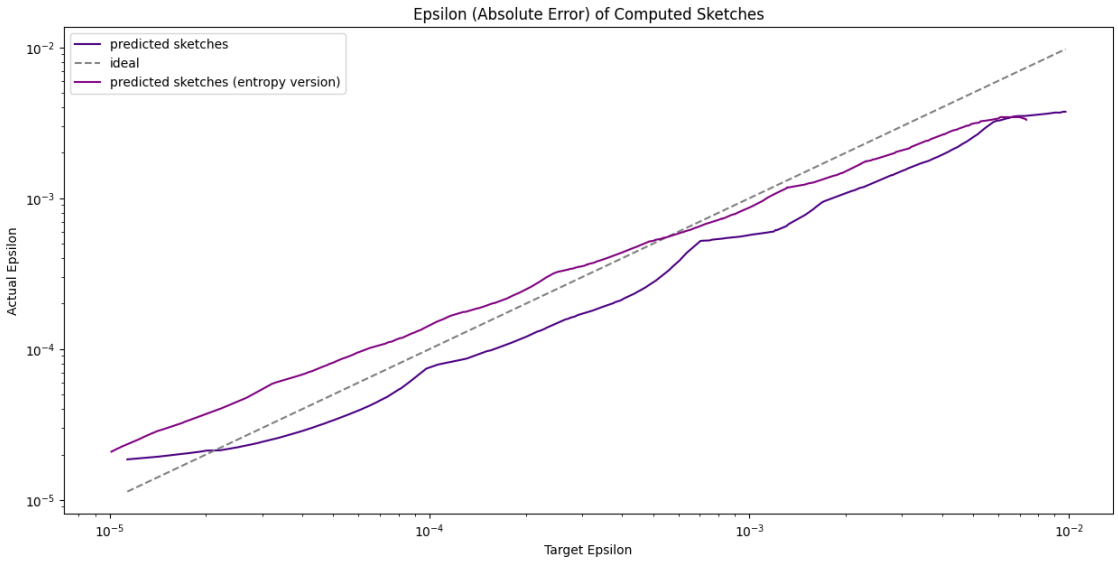

Figure 5: Comparison of entropy between small and large CMS. Larger sketches have higher entropy, indicating fewer hash collisions.

[6] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[7] Milind Srivastava, Shao-Tse Hung, Hun Namkung, Kate Ching-Ju Lin, Zaoxing Liu, and Vyas Sekar. Raising the level of abstraction for sketch-based network telemetry with sketchplan. In *Proceedings of the 2024 ACM on Internet Measurement Conference*, page 651–658, New York, NY, USA, 2024. Association for Computing Machinery.

[8] Haifeng Sun, Qun Huang, Jinbo Sun, Wei Wang, Jiaheng Li, Fuliang Li, Yungang Bao, Xin Yao, and Gong Zhang. AutoSketch: Automatic Sketch-Oriented compiler for query-driven network telemetry. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1551–1572, Santa Clara, CA, 2024. USENIX Association.

[9] Mads Zinkevich Michael Svenstrup, Nikolaj Hansen. Hash embeddings for efficient word representations. In *Advances in Neural Information Processing Systems (NIPS)*, pages 9343–9353, 2017.

[10] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, page 561–575, New York, NY, USA, 2018. Association for Computing Machinery.