Digital "Mad Men"
Training RNN as an Ad Copywriter

Paul Shved
pshved@stanford.edu

Christina Lin
chrlin@stanford.edu

Sridatta Thatipamala
sthatip@stanford.edu

Winter 2019
CS224n class at Stanford University
Google, Inc.

This is a **custom project**.

Mentor: Abi

Abstract

In this project, we aim to generate advertisements from positive product reviews and ad keywords. We train a "prototype-then-edit" neural model from [5] on a dataset of positive Amazon reviews and modify the content of the generated text using edit vectors drawn from a dataset of ad keywords. In contrast to the previous research, we produce the edit vector from the a secondary dataset, rather than the primary one on which the neural editor model is trained. We evaluate several variants of the model on fluency, relevance, and style. We find that the models are able to perform style transfer but are less successful in incorporating content keywords.

1 Introduction

Digital Advertisement is a growing 100+ billion dollar industry in the US [2]. Algorithms that can generate relevant advertisements automatically may significantly reduce advertiser costs and improve performance. However, not every text can serve as an ad; such texts should be: a) grammatically correct ("fluent"), b) relevant to the product they describe, and c) cast the product in a positive light.

We explore the use of neural networks to perform the creative work of generating ads. Many natural language processing tasks, such as question answering, machine translation, sentiment analysis, and parsing benefit heavily from using neural networks. Compared to these NLP tasks, generation of new text samples that fit certain requirements is less researched, and the available research demonstrates mixed results.

Current state of the art methods for natural language generation are based on recurrent neural language models. Some of the problems observed from such models include a tendency toward more generic and irrelevant sentences such as "I don't know" [7]. Instead of generating from scratch, another approach is to start with a high quality sentence and modify it to create a new sentence. One such model, presented in [8], transfers sentiment by replacing key phrases. Another model described in [5] samples a sentence and performs a random edit, which results in a fluent sentence of similar style that sounds like a paraphrase. Because the starting point is written by a human, these generated sentences are not biased towards shortness or vagueness [5]. It is, however, difficult to control the content of these sentences, and they end up similar to the training dataset.

In this paper, we apply the model from [5] to transfer the writing style of one dataset while drawing the content of the text from a second dataset. Specifically, we generate positive product reviews whose subjects are specified by ads keywords.

We find however that this way of constructing edit vectors for the decoder from [5] does not produce the desired edited sentences. The sentences generated are mostly grammatical, but the content of the keywords does not appear in the edited sentences. See section 4.2 for evaluation details.

2 Related Work

2.1 Neural approaches to NLG

The current state of the art method for style transfer is the variational auto-encoder model. Most of the current work in natural language generation relies on some form of autoencoders to achieve style transfer, and rest use the variational auto-encoder as a baseline for comparison.

In [6], Kingma and Welling leverage a Stochastic Gradient Variational Bayes (SGVB) estimator to optimize a neural network recognition model. Kingma and Welling describe a class of learning problems with directed probabilistic models, latent variables with intractable posterior distributions, and large datasets. They propose a reparameterization of the variational lower bound, which yields a lower bound estimator that can be optimized using standard stochastic gradient descent.

For style transfer in text in particular, the variational auto-encoder may be applied in the form of a variational auto-encoder seq2seq model. The encoder and decoder are both RNNs with GRUs. They are optimized using stochastic gradients obtained via backpropogation and the reparameterization described in Kingma and Welling.

We are particularly interested in an auto-encoder based approach to NLG that transfers style using a "prototype-then-edit" model, as it separates style transfer from content. In [5], Guu et. al attempt to develop a "unconditional generative model for text" which generates sentences similar to the sentences in their training corpus. In this model, the authors sample a "prototype" sentence x from the training corpus and sample a multidimensional "edit" vector z from a latent distribution. They then use a neural network, referred to as the "neural editor" to apply edit z to prototype x, thus generating sentence x'.

The paper demonstrates the performance of this model using two methods: perplexity and human evaluation. The neural editor lower perplexity than the other baseline models. While the perplexity provides insight into the performance of the language model, there are no quantitative metrics for how well the model accomplishes style transfer.

[5] is compelling because it learns a semantic space of edits to sentences. This technique of applying edits seems novel in the literature and could be useful in our task of text generation for ads.

2.2 Evaluation

NLP as a whole lacks standardized evaluation metrics, and the selection is even sparser for NLG. Many NLG work relies on perplexity and human evaluation, but we are interested in measuring our model's performance quantitatively.

In a 2017 paper, Fu et al propose the two novel evaluation metrics measure different aspects of style transfer: transfer strength and content preservation, respectively. Transfer strength measures how well the writing style is transferred from source to target, and it is measured using a sentiment classifier. Content preservation measures the consistency of the subject matter between two sentences, and it is computed using the cosine similarity between sentence embeddings. We adapt these ideas into the development of our own evaluation metrics.

3 Approach

In this work, we are using the neural model from [5] and apply it to a new dataset of Amazon Reviews. Additionally, we use the model trained this way to produce targeted edits, and evaluate the quality of the results. Our contribution includes:

- Preparation of the new dataset of Amazon Reviews to achieve Jaccard distance proximity (see 4.1).

- Implementation of the baselines and training of the baseline Language model using the code from [8] on the Amazon Reviews.
- Implementation of the Ad Generation algorithm 3.2, temperature smoothing.
- Implementation of evaluation metrics: Perplexity, Content Preservation, Transfer Strength.

For the model we train, its authors [5] sample a "prototype" sentence x' from the training corpus $\mathcal{X}$ and sample a 600-dimensional *"edit vector"* z from a latent distribution. Then, using an attention-based *"neural editor"*, the authors generate a new sentence x conditional on the edit vector. See example in table 2.

The likelihood of a sentence is, in the authors' model, is

$$p(x) = \sum_{x' \in \mathcal{X}} (p(x|x')p(x')) = \sum_{x' \in \mathcal{X}} \left(\mathbb{E}_{z \backsim p(z)} \left[p_{edit}(x|x', z) \right] p(x') \right) \tag{1}$$

The author mention that the model can not be trained on the log-likelihood $p(x)$ directly as it requires (a) summing over the large input dataset $\mathcal{X}$, and (b) the expectation over a uniform random sample $p(z)$ has no closed form and would be too slow to sample. The authors mitigate this by:

- Introducing an **inverse neural editor**: instead of computing $\mathbb{E}_{z \backsim p(z)}$, the authors construct the edit vector $z = q(x, x')$ from x and x' and add random noise to this vector. This produces a distribution $q(z|x, x')$.
- Maximizing the lower bound of $p(x|x')$ instead through Evidence Lower Bound over $q(z|x, x')$: $\log p(x|x') \geq \mathbb{E}_{z \backsim q(z|x,x')} + KL(q||p(z))$ where KL is the KL-divergence that penalizes discrepancy between $p(z)$ and q.

$q(x, x')$ consists of two concatenated sums or GloVe word vectors for the words added to x to produce x' and the words deleted. The authors make sure the specific method of adding random noise results in a differentiable negative log-likelihood.

The architecture for this model proposed in [5] is a seq2seq encoder and decoder with attention. The encoder is a 3 layer bidirectional LSTM with GLoVe embedding. The input to the decoder is a concatenation of the encoder output, and Wz where z is the edit vector and W is a linear layer. The decoder is a 3 layer unidirectional LSTM with attention over source sentence.

During data preparation, [5] generates input-output pairs from list of sentences based on Jaccard distance threshold between the sentences in the pair. For large datasets, this step is nontrivial, as $O(N^2)$ runtime is intractable. In our project, we implemented this algorithm and had to choose a different threshold; see section 4.1.

3.1 Training

During training, the authors sample an edit vector, which becomes the input to the decoder, using the following algorithm. For each $(x, x') \in \mathcal{X}$: compute $v_{inserted}$: sum of GLoVE embeddings of inserted words. Then compute $v_{deleted}$: sum of GLoVE embeddings deleted words. Then pass $v_{inserted}$ and $v_{deleted}$ through a linear layer and concatenate into v_{edit}, adding noise sampled from Von Mises-Fisher distribution (a distribution over points on a unit hypersphere)

Table 1: Sample Training Data

Prototype (x)	"i have used this headset for a couple of years now and it still works great"
Target (x')	"i have used this hundreds of times and it still works great"
Edit vector (z)	["headset" + "for" + "a" + "couple" + "of" + "years" "now"; "hundreds" + "of" + "times"] + noise

3.2 Text generation

After the edit model is trained, its decoder is used in the text generation algorithm, using a *dataset of keywords* $\mathcal{K}$). For each keyword phrase $k \in \mathcal{K}$

1. Generate the edit vector $z_k = [\sum_{word \in k} v_{word}; 0]$ using GLoVE embeddings for k
2. Let $G_k = \varnothing$. For each $x_p \in \mathcal{X}$
   (a) Use beam search and decoder to find x' that maximizes $p(x'|x_p, z_k)$
   (b) Let the loss on the decoded sentence (negative log-likelihood) be l
   (c) $G_k \leftarrow G_k \cup \{(x', l)\}$
3. Find bottom N elements in G_k that have the lowest loss

In order to increase diversity, we applied temperature when computing token probabilities in the beam search in step 2a, and evaluated its impact (see table 5).

The result of this algorithm on one example looks like this (Table 2). Note that the keywords do not actually appear in the decoded sentence; this behavior is not what we expected, but it is characteristic of this model. See section 5 for the details.

Table 2: Sample Text Generation algorithm result

Prototype (x)	"i have used this headset for a couple of years now and it still works great"
Keyword phrase (k)	"auto loans"
Edit vector (z_k)	["auto" + "loans"; 0] *(Note the 0 for the deletion part of z, and no noise)*
Decoded result k	**"i ve had this for a couple of years now"**
Loss l	0.33
Result we expected:	"i have used these auto loans for a couple of years now and it works"

3.3 Data preparation

We have trained the prototype-then-edit model over the Amazon product review corpus. Rather than using all pairs of sentences (x, x') from the training data, we only consider pairs with a high token overlap, as measured by Jaccard similarity.

To generate similar pairs efficiently, we used the datasketch Python library [9] to generate a locality-sensitive hash of each review sentences.

Specifically, we tokenized each sentence into words and computed a MinHash [1] over all token 3-grams. MinHash is a probabilistic algorithm to estimate the similarity of two sets. It outputs a hash value which can be used as a locality-sensitive hash for the set. That is, two sets with high degree of similarity are likely to map to the same MinHash. MinHash has the desirable property that $P[MinHash(x) = MinHash(x')] = JaccardSimilarity(x, x')$, therefore it can be used as a proxy to quickly find similar sets among a large dataset.

We used datasketch to index the hashes of all sentences and find collisions, i.e. similar sentences. We restricted our data to pairs of sentences with Jaccard similarity of at least 0.4.

Table 3: Example training data: sentences pairs of Jaccard similarity ≥ 0.4

Sentence 1	**Sentence 2**
Not bad for the price	Best model for the price.
It just cannot be beat!	This case just cannot be beat!
This camera I would highly recommend to my friends.	I would highly recommend this camera

3.4 Word Vectors and Vocabulary

The original neural editor code used word vectors that were trained on their Yelp review corpus. The Yelp vectors led to in many out-of-vocabulary words, because the distribution of Yelp reviews is very different than that of Amazon reviews. Since our model had a fixed vocabulary size of 30K, we trained our model using the GLoVE vectors of the 30K most common words in the Amazon corpus. This decreased the number of "UNK" tokens output by the model. A future extension of this project could fine-tune the GLoVE vectors using the Amazon review corpus, rather than taking a subset of the standard GLoVE vectors.

3.5 Baselines

The target algorithm is an encoder-decoder implementation for $P(x|x', z)$, where x' is the prototype sentence, x is the target sentence, and z is the edit vector generated from ad keywords. We use two baselines.

Baseline 1 ("Sampled Pairs") is similar to RETRIEVEONLY from [8]. For every input sentence, we retrieve the closest lexically similar input sentence from the original corpus (except the input sentence), as measured by Jaccard Distance.

Baseline 2 ("Concatenated Pairs") is used to compare our proposed Ad Generation algorithm vs non-neural simple approach. Given the keywords and a sentence from the test set, we replace the first word in the sentence with the keyword phrase. For example, the keywords "auto insurance price quotes" and the prototype "it is made very well and looks cool" form the baseline text, "auto insurance price quotes is made very well and looks cool" We expect this baseline achieve similarly high scores on Transfer Strength and Content Preservation, but lower scores on Perplexity than the target model.

4 Experiments

4.1 Data

We use the following datasets:

1. A dataset of 277,228 one-sentence positive Amazon product reviews from [8]. This is used to train the edit model.
2. A dataset of 814 keyword phrases [4] (1-2 words). We use this data to synthesize edit vectors and evaluate the prototype-then-edit model on various style transfer metrics.
3. A dataset of 200 randomly selected words from the 2000 most common words in the Amazon product reviews. This dataset was used in the same way as the dataset of keywords.

4.2 Evaluation Method

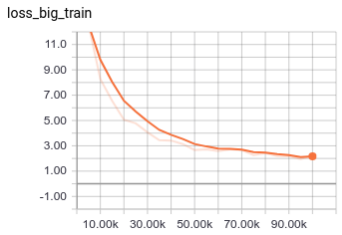
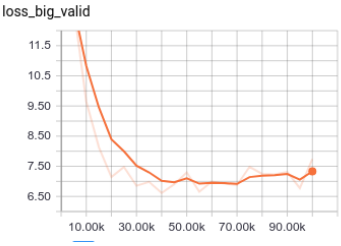
There is not yet a single standard quantitative metric to evaluation style transfer in text, so we use three separate metrics to measure fluency, content preservation, and transfer strength.

1. Fluency - We use perplexity of the edit model we've built on the generated sentences as a metric for Fluency. Due to time limitation, our original plan to use a general-purpose language model (such as BERT) is left for future work.
2. Content Preservation - We measure content preservation with cosine distance between source sentence embedding (original text) and target sentence embedding (generated review texts). The sentence embeddings are are a concatenation of the mean, minimum, and maximum vector values across words in a sentence. This metric was specified in [3], and similar methods are frequently used for evaluation of other NLP tasks without ground truth.
3. Style Transfer Strength - Trasnfer strength refers to how well the writing style has been transferred. In [3], style transfer strength is measured using a sentiment classifier trained on IMDB data. A generated sentence is considered of matching style if it is given the same classification as the corresponding source. The score for style transfer strength is given $\frac{N_{right}}{N_{total}}$, where N_{right} is the number of correctly classified generated sentences and N_{total} is the number of test data.

4.3 Experimental Details

The neural editor model was trained using the same parameters as in the original paper [5] (learning rate 0.001, batch size 128). However, we trained the model on the Amazon dataset only using $100,000$ iterations instead of $400,000$ since our dataset was smaller, and the model performance stopped improving on the validation set (Table 4). Total training time 7.5 hours on an nVidia RTX 2080 Ti GPU.

Table 4: Loss vs number of training iteratinos

Loss on training set	Loss on validation set
	

X axis: number of iterations; Y axis: loss

For the initial run, we sampled 500 keywords from the keyword dataset, and used 1024 randomly sampled prototypes. We batched the examples into batches of 512 items.

Our initial results did not demonstrate the desired performance. In order to explore how the edit model performs under various conditions, we have implemented the modifications to the input data and the decoder hyperparameters (see Table 5).

Table 5: Input Data and Hyperparameter experiment descriptions

Experiment name	Description
Editor: Ads Keywords	the keyword phases are sampled from the Ads Dataset, decoder temperature = 1.0
Editor: Ads Keywords (temperature = 3)	same as above but the decoder temperature = 3.0
Editor: Ads Keywords (30x)	apply the decoder again to the result of the decoder 30 times using the same edit vector constructed from the keywords (temperature = 1)
Editor: Amazon Keywords	Instead of sampling edit vectors from Keyword dataset, we use the words from the reviews themselves. Most of the Ad Keywords never appeared in the training set, and we hoped it could improve performance

4.4 Results

We evaluate several models and baselines with two style transfer metrics: classification, which measures transfer strength, and cosine similarity, which measures content preservation. Table 6 shows the metrics for two baselines and four models. All are trained on the Amazon reviews dataset. Three of the experiments perform edits with ads keywords, and one experiment performs edits with Amazon keywords.

As described in section 3.5 Baseline 1 is reflective of the training and test data. As a sanity check, we expect its scores to serve as a ceiling for our models' scores. Baseline 2 attains a nearly perfect content preservation score, since the source and target are almost identical, and it scores predictably lower on the perplexity. Content preservation is less than 1.00 since the first word of the review is removed. The classification score of 0.935 indicates that adding a few ad keywords without modifying the writing style changes the sentiment classification for some of the samples. A score of 0.935 can be considered very strong style transfer.

Our vanilla model, "Editor: Ads Keywords," performs much worse than Baseline 1 on transfer strength. This implies that after editing and inserting keywords, the generated text does not maintain the style of the original product review as well as Baseline 1. However, increasing the temperature greatly improves the model's transfer strength performance, although it decreases fluency. "Editor: Ads Keywords (temperature = 3" achieves better transfer strength than both Baselines. Taking into account noise in the metric, the model with increased temperature performs as well as Baseline 2, effectively completely maintaining the writing style. These conclusions are based on the assumption that sentiment classification is a perfect measure of transfer strength.

Overall, the perplexity on the decoded sentences is low since beam search aims to maximize the probability, which minimizes the perplexity. Perplexity of the edit model is comparable with the perplexity of a baseline language model from [5] that we retrained on the Amazon dataset, and achieved 5.78. In the project milestone, the higher value of the perplexity was a result of using a very narrow vocabulary; it was an error that we fixed.

All models perform approximately the same on content preservation, which is approximately on par with Baseline 1. This indicates that the generated text is similar in content to the source, but to fully analyze content preservation requires a more in-depth look at a few examples to better understand the cosine similarity metric. We perform this analysis in Section 5.

Table 6: Style transfer metrics for models trained and tested on positive Amazon reviews

Dataset	Transfer Strength	Content Preservation	Perplexity
Baseline 1: Sampled Pairs	0.860	0.927	-
Baseline 2: Concatenated Pairs	0.935	0.997	24.51
Editor: Ads Keywords	0.778	0.912	1.78
Editor: Ads Keywords (temperature = 3)	0.950	0.918	30.14
Editor: Ads Keywords (30x)	-	0.910	1.73
Editor: Amazon Keywords	0.744	0.915	1.30

The following metrics are used: Classification for Transfer Strength (higher is better), Cosine Similarity for Content Preservation (higher is better), and Perplexity (base 2) of the language model on the edited sentence for Fluency (lower is better).

5 Analysis

We compare the performances of our models to the one another and to the baselines by analyzing the metrics in Table 6. As detailed in Section 4.4, we find that adding edits from either the Ads or Amazon keywords dataset decreases the transfer strength, while increasing the temperature improves transfer strength. Additionally, the generated texts maintain content preservation at the same level as Baseline 1.

To better understand the meaning of the content preservation score, we qualitatively examine a few examples generated by the "Editor: Ads Keywords" model. Table 7 contains three representative examples from the ten generated texts with the highest content preservation scores, while Table 8 contains three representative examples from the ten generated texts with the lowest content preservation scores.

Based on Table 7, we observe that the content preservation score reflects more the content preservation between the prototype and the generated text rather than between the keywords and the generated text as intended. The bias is more severe when the prototype is a long sentence. None of the top score texts successfully incorporate the keywords into the generated text, yet they obtain high scores because the generated text is nearly identical to the prototype. The scoring system from [3] may work better for longer and more sentence-like keyword phrases, but for our purposes, it would have been useful to give a greater weight to the keywords and lesser weight to the prototype, rather than concatenating them together.

Table 8 gives examples of some common errors in our model. The first generated text is nonsensical and repetitive, which may reflect some weakness in the language model related to the word "utensils." Note that this example attains a content preservation score of 0.770, suggesting that a score around 0.7-0.8 should be considered very low quality.

The second generated text is "but with the <unk> that comes with the back," which is grammatically correct and contains an <unk> where a noun, such as the keywords "home loans refinance," could potentially be inserted. Such examples are some of the more promising results from our model. However, this example has a low content preservation score because it is very different from the prototype and also does not contain the keywords.

The third generated text, "in fact , <unk> if you just want to try it , <unk> not an issue," contains several <unk> tokens that follow punctuation marks, and if they are removed, the sentence is nearly

fluent. This type of error recurs in many examples from our model, which may indicate that there is a related bug in the model.

Based on our examination of a few examples, we conclude that a high content preservation score may not indicate that the content of the keywords was successfully transferred to the generated text, and that a "high" score is likely around 0.9 or higher. Quantitative metrics for NLP and particularly NLG are inexact and must be considered alongside human evaluation.

Table 7: Examples with highest content preservation scores, generated by a model trained on Amazon reviews with ads keywords

Keywords	refinancing
Prototype	the cheap screen on my ipod touch gets sparkly after a good wipe , the steinheil gets more smeary
Generated Text	the cheap screen on my ipod touch gets sparkly after a good wipe , <unk> the steinheil gets more wipe
Preservation	0.987
Keywords	sky uk
Prototype	you can search google for this kind of side effects , i have found some articles .
Generated Text	google search and you can search for this kind of side effects , <unk> i bought some articlee
Preservation	0.984
Keywords	earn so
Prototype	if your pain is caused by bending your wrist downward too much , this is the brace for you
Generated Text	so if your pain is caused by bending your wrist into bending your wrist your wrist granddaughters wrist d
Preservation	0.982

Table 8: Examples with lowest content preservation scores, generated by a model trained on Amazon reviews with ads keywords

Keywords	**Prototype**	**Generated Text**	**Content Preservation**
fargo refinance	the set fulfills my range of needs while adhering to budget constraints	utensils utensils utensils utensils and utensils utensils utensils utensils kong and utensils utensils...	0.770
home loans refinance	this is the next best thing to a cabinet trash can that comes with the cabinets .	but with the <unk> that comes with the back .	0.826
law lexington	see how my cat likes it , how it washes , if it discolors , chips , or rusts , etcetera .	in fact , <unk> if you just want to try it , <unk> not an issue .	0.828

6 Conclusion

6.1 Findings

In conclusion, we have successfully trained a neural editor model to output sentences that are similar to Amazon reviews. We also aimed to craft edit vectors that control the output sentences such that they contain desired keywords. We evaluated the generated texts with both quantitative and qualitative analyses. Our model does not perform as expected in this second task involving desired keywords. The generated output sentences do not frequently contain the keywords from the edit vector.

There are many potential reasons for this. The style and vocabulary of advertisements are very different to that of Amazon reviews. Since the neural editor is ultimately a conditional language model, it may not assign high probability to such sentences. The model was also trained by framing sentence editing as a translation task. Therefore, it may have learned to output sentences without much regard to the edit vector.

Natural language generation is still a developing field, and text style transfer is even more nascent. In this project, we tested one approach to textual style transfer and found its strengths and limitations. We believe this can contribute to future attempts.

6.2 Future Work

Due to the time limitation on this project, we found many more opportunities than we had time to explore. We recommend future work on this topic to consider:

1. Improve loss function to take into account transfer of content from edit vector
2. Use Transformer-based architectures to apply attention over the edit vector words as well as over the prototype sentence
3. Modify Beam search to prefer candidates that contain edit keywords over probable-but-boring sentences
4. Perform detailed analysis of our key metrics: classification and cosine preservation. Apply regression to compare their values against human evaluation. We suspect that these measures are not linearly correleated with style transfer quality.

Acknowledgements

We would like to thank Professor Chris Manning for teaching a fantastic class and Abigail See for her invaluable advice and guidance on this project.

References

[1] A. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, SEQUENCES '97, pages 21–, Washington, DC, USA, 1997. IEEE Computer Society.

[2] eMarketer Corey McNair. Us ad spending 2018, 2018.

[3] Zhenxin Fu, Xiaoye Tan, Nanyun Peng, Dongyan Zhao, and Rui Yan. Style transfer in text: Exploration and evaluation. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[4] University College London Google. Open advertising dataset, 2014.

[5] Kelvin Guu, Tatsunori B. Hashimoto, Yonatan Oren, and Percy Liang. Generating sentences by editing prototypes. *CoRR*, abs/1709.08878, 2017.

[6] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[7] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models. *arXiv preprint arXiv:1510.03055*, 2015.

[8] Juncen Li, Robin Jia, He He, and Percy Liang. Delete, retrieve, generate: A simple approach to sentiment and style transfer. *CoRR*, abs/1804.06437, 2018.

[9] Erkang (Eric) Zhu. datasketch. [Online; accessed <today>].