# An Exploratory Analysis of Fake News Detection on Adaptive Learned Bloom Filters

COMP 480: Final Project

Abdul Abouzeid, Alexander Xiong

Mentor: Zhaozhuo Xu

# Introduction

We implemented four machine learning-driven bloom filters and examined the accuracy-resource tradeoffs. The following set of algorithms: bloom filters, learned bloom filters, sandwiched learned bloom filters, adaptive bloom filters, and disjoint adaptive bloom filters were compared and contrasted.[1] More specifically, we wanted to focus on conducting further exploration into the reproducibility and extensibility of the previous work. Hence, we followed the process outlined in the paper as well as validated the results presented.

We chose two new datasets, beyond the ones used in the paper, for tackling the problem presented in the testing purposes. Under discussion with our mentor Zhaozhuo Xu, we settled on the theme of fake news detection. Hence, both datasets applied in this project are related to this category with the focus of one on mimicking the size constraints posed in the previous work and the other tackling large memory usage.

Initially, we focused on the reproducibility of previous results but we also examined the false positive rate vs. queries per second and the effects of hyper-parameters for learning on the aforementioned responses, across all datasets.

# Background

As the focus of our report is on the applicational aspect of this project and verification of results, we will not present in-depth on the specifics of each bloom filter variation. The details as well as the mathematics regarding each variant can be found in the original paper.[1]

## Bloom Filters[2]

A Bloom Filter expands on the idea of combining a bitmap with universal hashing. Bloom Filters solve the problem of efficient membership query (or a cache) because caches still work with false positives (since you can incur an extra cost to actually check if it is positive). The guarantee of no false negatives is really helpful for caching problems because we want to ensure low memory constraint, fast access, and accuracy.

## Learned Bloom Filters[1]

A Learned Bloom Filter is pre-trained to classify whether any specific query belongs to the set of keys. By using binary training to set a numeric threshold to determine the previous classification, learned bloom filters can decrease the number of keys to examine by the bloom filter. Thus, we now define false positives as data points incorrectly classified using the binary classifier and false positives from the bloom filter. The tradeoff is that higher positive keys identified by the classifier means a lower FPR from the bloom filter due to a smaller chance of collisions. Since we also want to minimize the number of positives, higher threshold values are better, resulting in more keys to insert into the bloom filter.

## Sandwiched Learned Bloom Filters[1]

A Sandwiched Learned Bloom Filter is where an initial filter is added before the binary classifier to improve the false positive rate. To implement the sandwiched learned bloom filter, the parameters are optimized, selecting $\tau$ and calculating the corresponding initial and backup filter size.[2] However, in the scenario where the optimal backup filter size is much larger than the

budget, the sandwiched LBF does not need an initial filter and reduces to a standard learned bloom filter.

## Adaptive Bloom Filters[1]

An Adaptive Bloom Filter adjusts the number of hash functions differently in different regions to adjust the FPR adaptively.

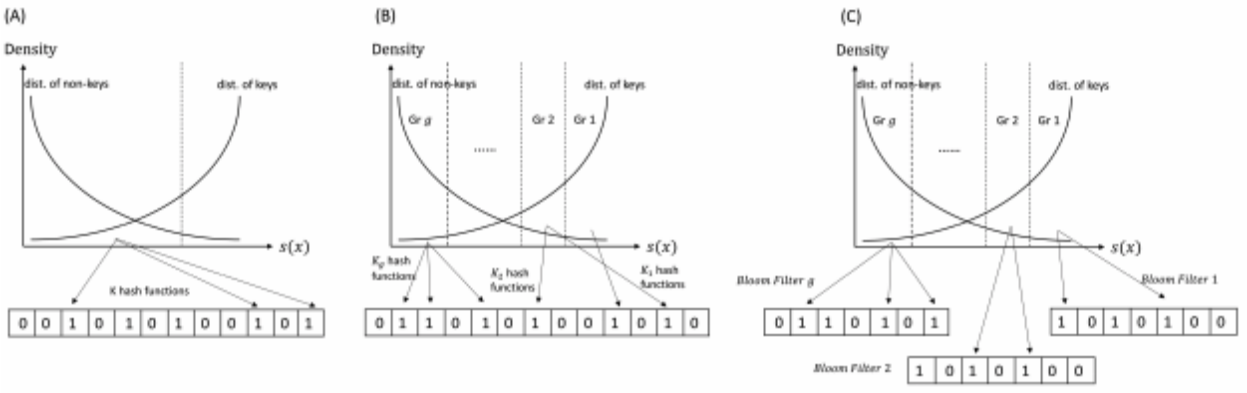


Fig 1. Panels A-C show the structure of LBF, Ada-BF and disjoint Ada-BF, respectively.[1]

To avoid overloading the bit array, we only increase the number of hash functions for groups with a large number of keys, while decreasing for groups with a small number of keys. This adaptation gives flexibility in selectively tuning each region of the bloom filter, thus, improving the false positive rate of the bloom filter as a whole. When the adaptive bloom filter only divides the queries into two groups, Ada-BF reduces to the LBF.

## Disjoint Adaptive Bloom Filters[1]

A Disjoint Adaptive Bloom Filter allocates variable memory Bloom filters to each region. As we can see from Figure 1. above, this means that we need to use a tuning strategy in order to calculate the optimal allocation of memory to each region's bloom filter. This strategy reduces the overall false positive rate by reducing the false positive rate of regions that are dominated by non-keys.

# Methodology

## Data Gathering

We started by preparing a dataset that has appropriately classified real news sources and fake news sources.[3,4,5,6] The aggregated dataset spans a variety of collection methods and publishers, containing 167,156 real headlines/articles and 38,918 fake headlines/articles.

## Random Forest Training

Although the specifics of the random forest model building were not attached in the code nor further elaborated on in the paper, we mimicked a similar approach to determine the "fakeness score" of each article. Opposed to the classification features available in the malicious url dataset such as "host name length", "path length", and "length of top level domain", we use language processing libraries to tokenize the headlines and article bodies. We then apply the TF-IDF vectorizer to appropriately process input for the random forest.

The pre-processing consisted of reducing the data to textual data, applying the Porter Stemmer algorithm, and then removing anystop words across aggregated NLP stop words. We vectorize the titles and text using TF-IDF vectorization and train the random forest classifier on the processed data. TF-IDF is a text vectorizer that combines the term frequency within documents to the inverse document frequency.[7] Two datasets were generated.

1. We wanted to generate a dataset that mimicked the size constraints imposed on the previous work model. The previous costs were 146 KB and 136 KB for the malicious url and malware data, respectively. Thus, we found that 20 leaf nodes and 40 decision trees resulted in the closest memory usage of 155 KB.
2. Another focus was to benchmark the minimized memory dataset against a non-parameterized random forest to distinguish the extent of the memory-to-accuracy tradeoff. A full random forest contains 100 decision trees with an unbounded number of leaf nodes. For comparison, this model resulted in a memory usage of 384 MB.

The testing accuracy of the two models were 81.06% for the mimic model and 90.17% for the full model.

By applying the models to the entirety of the training and testing dataset, we can obtain score measurements that reflect the score measurements used in the paper. Now, we can pipeline the new data into the bloom filter algorithms proposed in the paper.

# Results

## Histogram Score Distribution Partitions

Keys represent malicious urls, viral files, or fake news. Non-Keys represent the inverse of each, respectively.

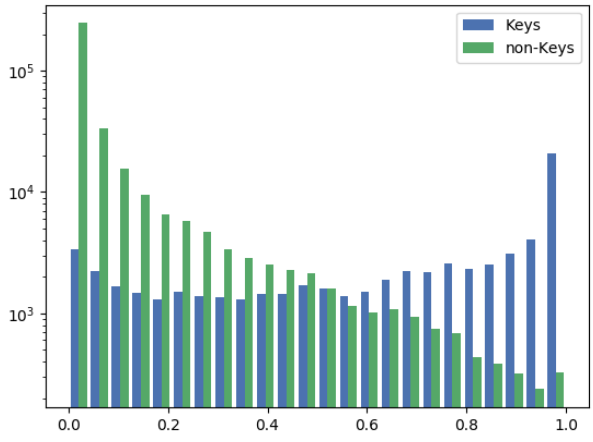


Fig 2a. Malicious vs. Non-Malicious urls

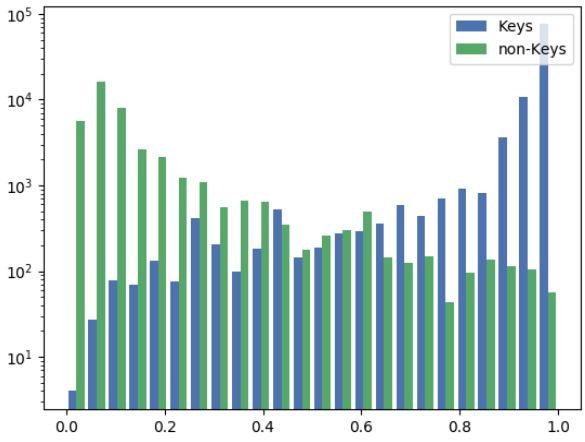


Fig 2b. Viral vs. Benign files

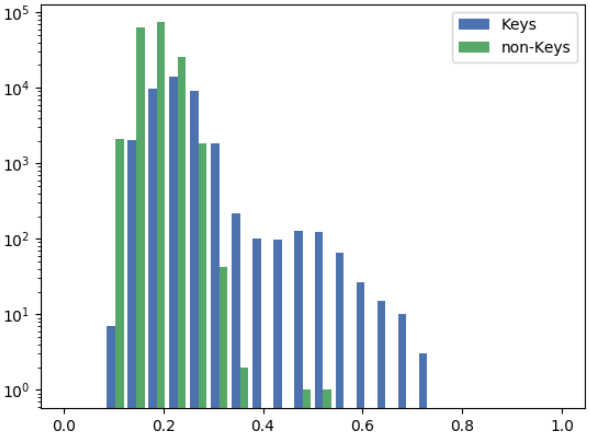


Fig 2c. Fake vs. Real news articles

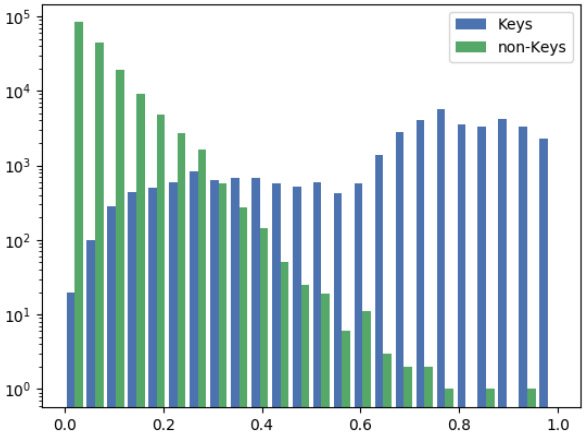


Fig 2d. Fake vs. Real news articles

We observe that while the previous datasets used in the paper could be trained with a smaller model, the fake news dataset struggles to produce a key vs. non-key distribution under such size constraint. This will help demonstrate how the bloom filters perform under a weakly vs. strongly trained model.

## Queries per second vs. FP rate

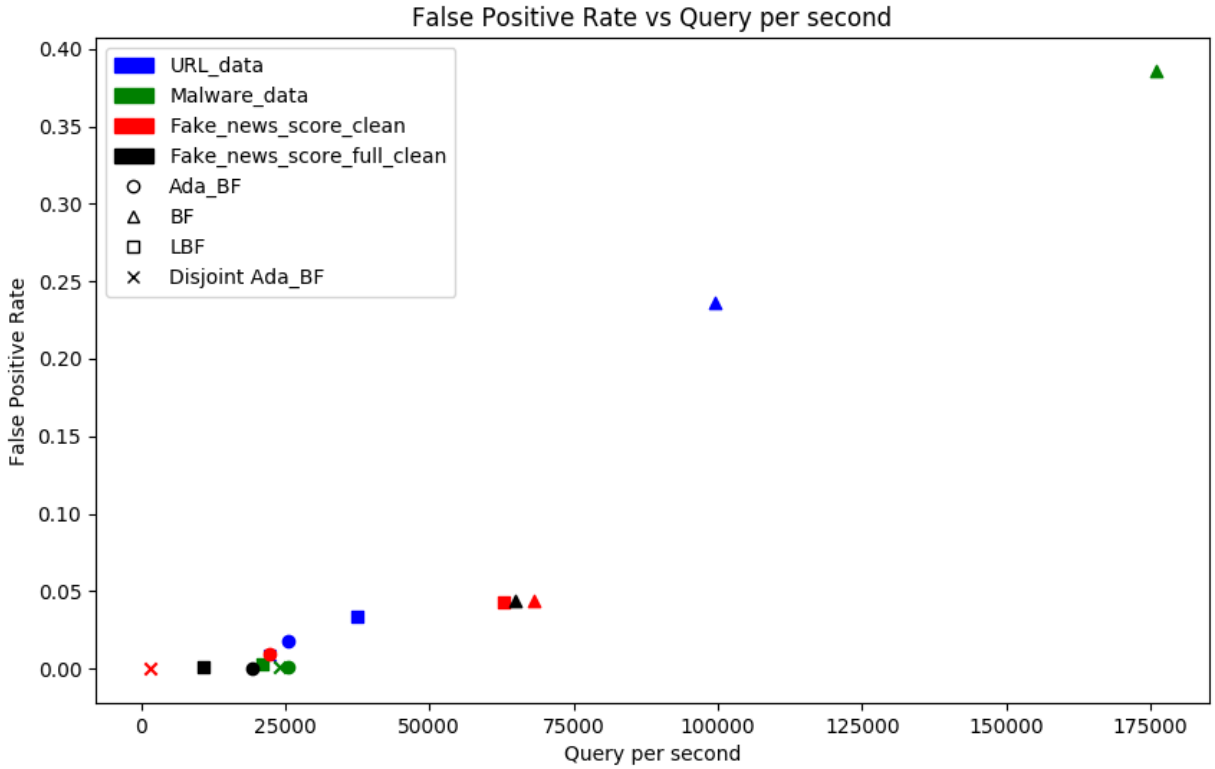


Fig 3. Queries per second vs. FP rate over all datasets and bloom filter variants

This plot demonstrates the relationship between false positive rates and the speed of bloom filter queries over different bloom filter variants tested over the various datasets. We see that the regular bloom filter has the largest average queries per second while the other bloom filter variants are more dataset dependent. Although not clearly visible in the plot above (the url dataset's disjoint adaptive bloom filter data point is at effectively the same location as the malware dataset's disjoint adaptive bloom filter data point), another visible trend is that

employing the disjoint adaptive bloom filter is the most consistent bloom filter variant in optimally minimizing the false positive rate over various datasets. Compared to the adaptive bloom filter, the disjoint adaptive bloom filter is much more stable and produces results below 1% whereas the adaptive bloom filter resides around a 2% false positive rate. While the learned bloom filter, on average, is only around a 3% false positive rate, we clearly see the tradeoff between queries per second and false positive rate between the learned bloom filters and adaptive bloom filters.

## Memory Budget vs. FP rate

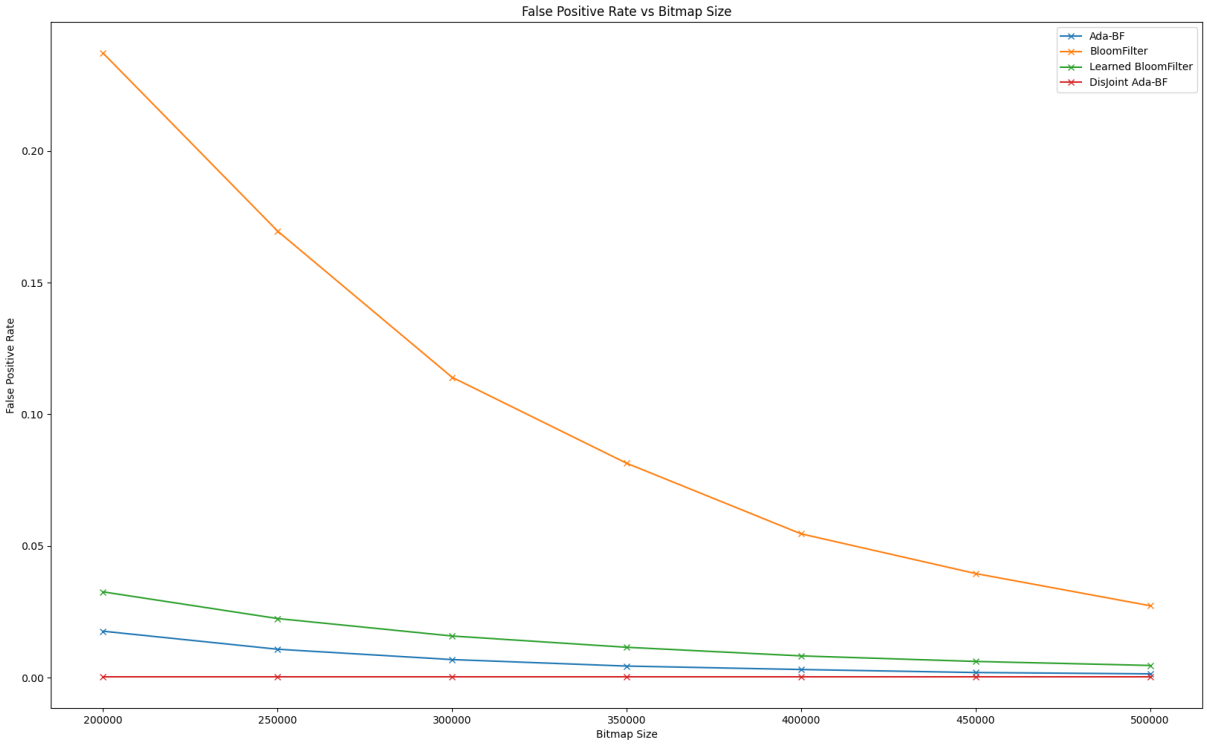


Fig 4a. Malicious vs. Non-Malicious urls Memory Budget vs. FP rate

We observe a similar trend as seen in Fig. 4a of the previous work. As expected, false positive rates decline as the memory budget increases due to the decrease in collision likelihood.

Although we do not observe the same nuances in the trends, for example, the previous work had a sharp decline at 400 KB, this can be attributed to differences between trials.

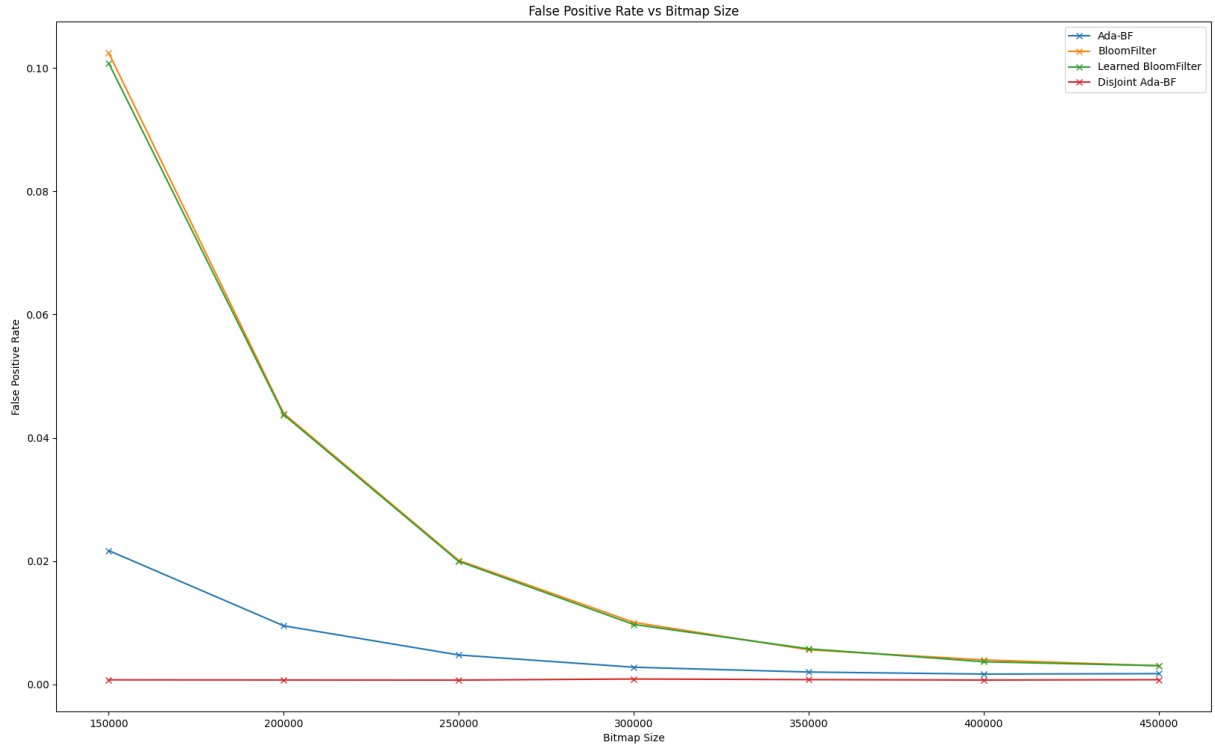


Fig 4b. Fake News vs. Real News Memory Budget vs. FP rate

We observe a similar trend as seen before. As expected, false positive rates decline as the memory budget increases due to the decrease in collision likelihood. The disjoint adaptive bloom filter had the best false positive rate over increasing bitmap size, followed by the adaptive bloom filter, then the learned bloom filter and regular bloom filter. Over a large bitmap size, the accuracy tradeoff is negligible but there are clear differences when comparing for smaller bitmap sizes such as when the bitmap size equals 150 KB. An interesting observation is that the learned bloom filter in this case offered no accuracy increase compared to a basic bloom filter. This may be due to the fact that the model used to obtain the scores for learning was unable to offer any clear differentiation between the categories leading to no false positive gain in the bloom filter

structure. We will further observe this phenomenon as we can compare Fig 4b. to Fig 4c. to determine the effect of the random forest training.

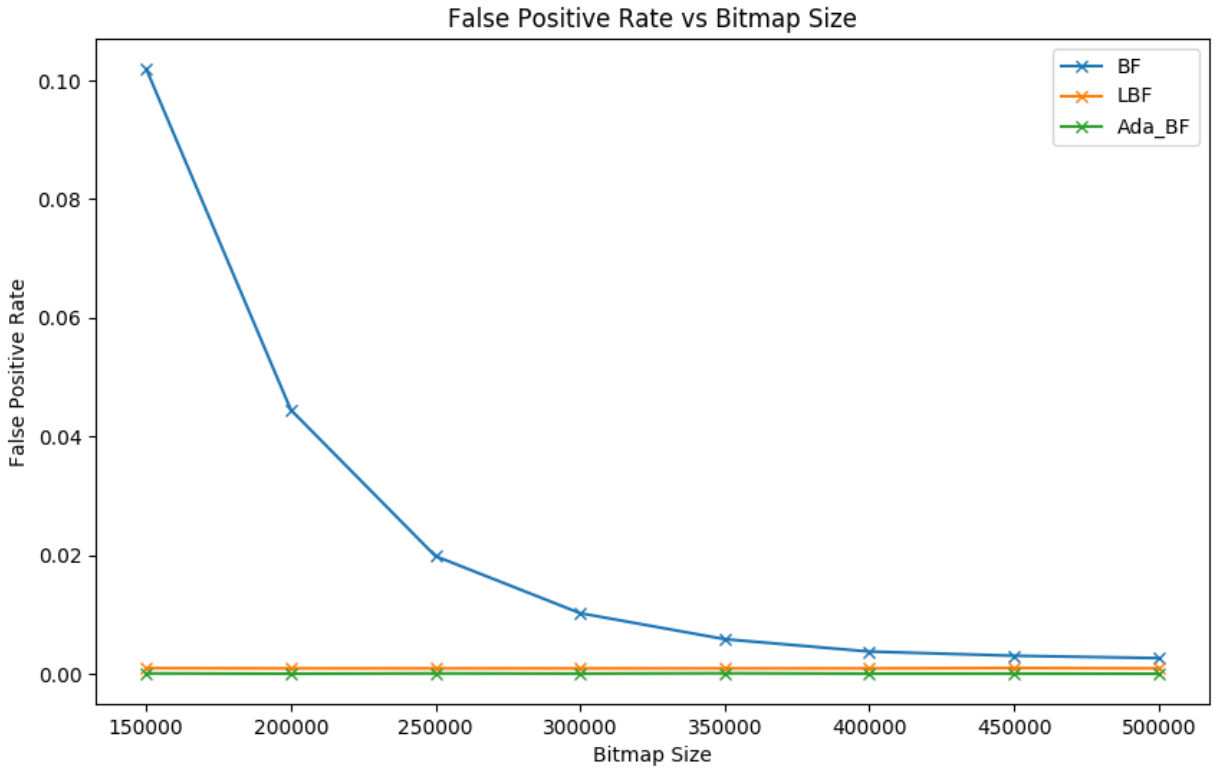


Fig 4c. Fake News vs. Real News: Fully trained model, Memory Budget vs. FP rate

This result is consistent with what is observed in Fig 4a. We see that the learned bloom filter matches the adaptive bloom filter (the adaptive bloom filter has a slightly better false positive rate) while the false positive rate of the regular bloom filter drops as the bitmap size increases. Specifically, we observe that the false positive rate tradeoff between the adaptive bloom filter and the learned bloom filter is minimal and that we believe the disjoint adaptive bloom filter will match this trend. This will be expanded on further in the discussion but we were unable to obtain and plot the disjoint adaptive bloom filter results.

### Hyper-parameter Expansion

We tried increasing the hyper parameters of the minimum groups to 6, the maximum groups to 20, the minimum c value to 1.0, and the maximum c value to 3.0 to compare false positive rates and query times for the adaptive bloom filter and the disjoint-adaptive bloom filter; however, we ran into problems with how the respective bloom filters were constructed. Further exploration would be to isolate these problems to help build a better generalized adaptive and disjoin-adaptive bloom filter.

## Discussion

We attempted applying the large model to the disjoint adaptive bloom filter but obtained a "Killed: 9" output due to large memory usage. Thus, in our experimentation, we were bounded by the machine when overloading the memory usage of the bloom filter. Further experiments can be performed by adjusting the input for the disjoint adaptive bloom filter to take in a scalar value of additional memory usage by the model rather than the model itself. That way, variable memory usage by the model can be tested.

Furthermore, we also planned on conducting a deeper analysis into the disjoint adaptive bloom filter but due to an unforeseen restriction regarding the execution time of the disjoint adaptive bloom filter, we had to cut this aspect as it would not finish before the deadline (ran for multiple days without completion). Specific examination should go into this chunk of code

```
while abs(sum(R) - R_sum) > 200:
     if (sum(R) > R_sum):
         R[non_empty_ix] = R[non_empty_ix] - int((0.5 * R_sum) * (0.5) ** kk + 1)
     else:
         R[non_empty_ix] = R[non_empty_ix] + int((0.5 * R_sum) * (0.5) ** kk + 1)
    R[non_empty_ix:] = R_size(count_key[non_empty_ix:-1],
count_nonkey[non_empty_ix:-1], R[non_empty_ix])
```

```
    if int((0.5 * R_sum) * (0.5) ** kk + 1) == 1:
        Break
    kk += 1
```

Only the fake news score full clean dataset triggers this issue so further examination in the algorithm design would be crucial to build a generalized application for this bloom filter variant. Future exploration can also be in optimizing the disjoint adaptive bloom filter for execution speed while preserving the memory optimization structure (ie. more careful hyperparameter selection when optimizing).

# Conclusion

The scope of this project was limited to experimental replication and application of new data. As discussed, we were able to find flaws in the wide-scale usability of this application towards different datasets. We achieved our goals but would definitely look forward to expanding the technical aspects of this project as well in the future.

Another direction that may be interesting to pursue is following up with the future works of detecting near matches for a fake news detection application using learned, distance sensitive bloom filters.[8]

# References


1. Adaptive Learned Bloom Filter (Ada-BF): Efficient Utilization of the Classifier
    a. https://arxiv.org/pdf/1910.09131.pdf
2. A Model for Learned Bloom Filters and Optimizing by Sandwiching
    a. https://papers.nips.cc/paper/2018/file/0f49c89d1e7298bb9930789c8ed59d48-Paper.pdf
3. BS Detector Fake News Dataset
    a. https://www.kaggle.com/mrisdal/fake-news
4. Fake and Real News Dataset
    a. https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset
5. Fake or Real News Dataset
    a. https://bitbucket.org/WiZar-D/fake_real_dataset/downloads/
6. All the News, Reals News Dataset
    a. https://www.kaggle.com/snapcrack/all-the-news?select=articles3.csv
7. TFIDF
    a. http://www.tfidf.com/
8. Distance Sensitive Bloom Filters
    a. https://www.eecs.harvard.edu/~michaelm/postscripts/alenex2006.pdf


# Appendix

Our code can be found here: https://github.com/awx1/comp480-final in the Ada-Bf-master folder.

Important files:

Driver.py

- Generates the false positive rate vs. queries per second over the different bloom filters and datasets.

Driver2.py

- Generates the false positive rate vs. bitmap size plots over the different bloom filters and datasets.
- Used for validating results from previous work.