USENIX'23 Artifact Appendix
AutoFR: Automated Filter Rule Generation for Adblocking

Hieu Le[*] Salma Elmalaki[*] Athina Markopoulou[*] Zubair Shafiq[†]

[*]University of California, Irvine [†]University of California, Davis

A Artifact Appendix

Adblocking relies on filter lists, which are manually curated and maintained by a community of filter list authors. We introduce AutoFR, a reinforcement learning (RL) framework to fully automate the process of filter rule creation and evaluation for sites of interest. Examples of filter rules are in Table 1. AutoFR is the first to balance the trade-off between blocking ads *vs.* avoiding visual breakage. The user gives AutoFR inputs (*e.g.,* the website to generate rules for, and breakage tolerance threshold w) to AutoFR. It will run our RL algorithm based on multi-arm bandits and generate filter rules that block ads while adhering to the given w threshold. This appendix details how to access our artifact (implementation of AutoFR and our dataset) and how to use and evaluate it.

A.1 Abstract

Our artifact includes the following. First, we open-source an implementation of the AutoFR framework on GitHub. Second, we provide our dataset of collected site snapshots on the Top–5K sites, which can be utilized to reproduce the filter rules we created or explore other algorithms to generate rules.

AutoFR's implementation follows Algorithm 1 and is illustrated in Fig. 4. Notably, it uses site snapshots (Fig. 5 and Sec 4.1), which are graph representations of how a site is loaded. We use them offline to run the reinforcement learning logic, which removes the bottleneck of waiting for a site to load during every visit. See Fig. 5 for more details.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

At its core, AutoFR visits websites automatically and creates filter rules that block ads with minimal visual breakage. Thus, there may be security issues if the user gives AutoFR a malicious site to visit. We advise testing AutoFR on sites that the user trusts. In terms of privacy, if AutoFR is used on a personal machine, websites may fingerprint or track the utilization of AutoFR.

A.2.2 How to access

GitHub: The repository is listed at https://github.com/UCI-Networking-Group/AutoFR/tree/artifact-review. It provides a detailed README.md on how to use AutoFR. The rest of this appendix will refer to https://github.com/UCI-Networking-Group/AutoFR/tree/artifact-review.

Dataset: The dataset and its detailed description are available at https://athinagroup.eng.uci.edu/projects/ats-on-the-web/autofr-dataset/. In summary, the dataset contains 1042 zip files, one per-site. Each zip file includes the raw collected data of outgoing HTTP requests, AdGraphs, annotated site snapshots, the action space, filter rules, and more. This matches Table 2. This includes a *"Top5k_rules.csv"* file that shows all the filter rules created within each zip file. Users must sign a consent form (at the bottom of the web page) before accessing the dataset. For *artifact reviewers*, we provide the direct Google Drive link to the dataset within a hotcrp comment.

A.2.3 Hardware dependencies

AutoFR was evaluated using Amazon EC2 instance m5.2xlarge, which has 8 cores, 32 GiB of memory, 35 GiB of storage, and up to 10 Gbps of network bandwidth. We recommend something similar, going as low as 16 GiB of memory with 20 GiB of storage. Our repository will provide a Dockerfile for easy setup.

Limitations. Currently, we do not support the running of AutoFR on M1 MacBooks (ongoing work to support it).

A.2.4 Software dependencies

AutoFR has been tested on a Debian 5.10 server (university) and Ubuntu 18.04.6 LTS (AWS EC2). Implementing the framework includes using several Python libraries, browser extensions, and prior work. The majority of the dependencies will be encapsulated in a Dockerfile. We list the major ones below and refer to the README.md of https://github.com/UCI-Networking-Group/AutoFR/tree/artifact-review for details.

Core Dependencies (Must Haves):

- Python 3.6+, git, pip3, virtualenv (or conda), docker
- If necessary, install with:
  1. sudo apt-get install git python3 python3-dev python3-pip
  2. pip3 install virtualenv
  3. We defer the docker installation to https://docs.docker.com/engine/install/debian/.

Dependencies (within Dockerfile):

- Python 3.6+: tldextract, networkx, adblockparser, pandas, numpy, selenium
- NodeJS: Ad Highlighter, Adblock Plus (browser extensions)
- C++: AdGraph (instrumented chromium)

A.2.5 Benchmarks

None

A.3 Set-up

For easy copy and paste of commands, we recommend using the https://github.com/UCI-Networking-Group/AutoFR/tree/artifact-review#setup.

A.3.1 Installation

1. Git clone our AutoFR repository (see Sec. A.2.2). The rest of the instructions assume you are in the project directory using a terminal window.
2. For artifact reviewers: "git checkout artifact-review"
3. git submodule update --init --recursive
4. Create a python virtual environment and activate it. We recommend using "virtualenv".
   (a) virtualenv --python=python3 [/save-path/autofrenv]
   (b) source [/save-path/autofrenv]/bin/activate
5. Install AutoFR:
   (a) pip3 install -e .
   (b) Make sure to have the period at the end of the command.
   (c) mkdir temp_graphs; mkdir -p data/output/
6. Build the docker container that AutoFR leverages:
   (a) docker build -t flg-ad-highlighter-adgraph --build-arg USER_ID=$(id -u) --build-arg GROUP_ID=$(id -g) -f framework-with-ad-highlighter/DockerAdgraphfile .
   (b) Make sure to have the period at the end of the command. This should run without any errors.
7. Done: You are now ready to run AutoFR.

A.3.2 Basic Test

Ensure you are in the project directory with a terminal window and your virtualenv activated as instructed in Sec. A.3.1.

1. Test whether your AutoFR environment has the necessary dependencies:
   (a) python scripts/autofr_controlled.py
   (b) The above command should print out a help message on how to use the script without errors.
2. View the docker image that you created:
   (a) docker image ls | grep flg-ad-highlighter-adgraph
   (b) The above command should print out the docker image called "flg-ad-highlighter-adgraph" with additional information such as its size.

A.4 Evaluation workflow

Disclaimer. As noted in our paper, the web changes naturally. AutoFR is only as good as its components. Thus, if a site does not serve ads that Ad Highlighter can detect or use obfuscation techniques, then AutoFR may not be able to generate rules for the given site. See Sec. 5.3.4 and 4.3. There may be other factors, such as w being too high to generate rules for, etc... Over time, AutoFR will improve as we maintain it, but we cannot guarantee that it will work on every website.

A.4.1 Major Claims

(C1): *Create Filter Rules:* Given inputs such as a website and hyper-parameters like the w threshold (breakage tolerance), AutoFR will generate filter rules that block ads with breakage that is within the w threshold. This is proven by the experiment (**E1**). Our results for the Top–5K sites are reported in Sec. 5.1, Table 2, Fig. 6(a-b), and Table 3 column 2. The w threshold ranges from 0–1; higher values mean the user wants to avoid breakage at the expense of not finding any filter rules that meet that criterion. In our paper, we use $w = 0.9$. See Sec. 3 and particularly 3.2.2 for details about our formulation.

(C2): *Reproducibililty:* Researchers can reproduce our results (*i.e.,* generate the same rules) by utilizing our collected site snapshots (provided in our dataset). This assumes the inputs to AutoFR are identical, and the same changeset/version of AutoFR is utilized (Sec. A.2.2). This is proven by the experiment (**E2**). Site snapshots are described in Sec. 4.1, Fig. 5, and Table 2. See further discussion in Sec. A.5.

A.4.2 Experiments

(E1): *Create Filter Rules: [10 human-minutes + compute-minutes vary on server + storage varies on site] × per-site*. See (**C1**) for more information.
How to: Run AutoFR to generate rules for a few given sites. Results will vary based on context, such as on the site, location, and given inputs. Repeat the below for a few sites. We recommend cricbuzz.com, yahoo.com, and sohu.com.
Preparation: Follow the instructions in Sec. A.3.1.
Execution: 1. Follow the below:

2. python scripts/autofr_controlled.py --site_url "https://cricbuzz.com" --chunk_threshold 6
3. The chunk size affects the number of docker instances spawned to visit the given website. Based on Sec. A.2.3, we recommend 6. If you have fewer cores, then decrease the chunk_threshold.

Results: 1. Follow the below. Directories given are relative to the project directory:

2. The terminal will display the rules that are outputted.
3. Go to directory "data/output/" to see the raw collected data, such as the outgoing HTTP requests, AdGraphs, and site snapshots.
4. Go to "temp_graphs" to see the outputted filter rules and other information.
5. Full explanation of the output is explained in our README: https://github.com/UCI-Networking-Group/AutoFR/tree/artifact-review#understanding-the-output.

Test the Rules In-the-Wild (optional): 1. Follow the below if you want to try the rules in your browser.

2. Install an adblocker, like Adblock Plus, into your browser (instructions depend on your browser).
3. Turn the rules given by AutoFR into per-site rules. For each rule, append the site it was created for. For instance, if the rule is ||*doubleclick.net*^ for the site cricbuzz.com, then change it to ||*doubleclick.net*^$domain=cricbuzz.com.
4. Configure the extension by going to its settings. Turn off all filter lists. Add in custom rules from the previous step. See https://help.adblockplus.org/hc/en-us/articles/360062859913-Add-a-custom-filter.
5. Refresh the site to see if ads are blocked. Note if there is any visual breakage.
6. Remember to undo the changes if you use the adblocker personally.

(E2): *Reproducibility: [5 human-minutes + 2 compute-minutes + no storage] × per-site*. See (**C2**) for more information. This is completely offline.
How to: Run AutoFR with existing site snapshots to reproduce the results. Repeat the below for a few sites. We recommend cricbuzz.com, yahoo.com, and sohu.com.
Preparation: Download the "Top5K_rules.csv" file. Open it and choose a zip file to download, described in Sec. A.2.2. Here we assume you chose AutoFRGEval_www.cricbuzz.com_ad3dce7b.zip. Unzip the file. Then, follow the instructions in Sec. A.3.1.
Execution: 1. Follow the below:

2. python scripts/autofr_use_snapshots.py --site_url "https://www.cricbuzz.com/" --snapshot_dir [zip name]/[Snapshots directory]
3. A full example is provided at step 7: https://github.com/UCI-Networking-Group/AutoFR/tree/artifact-review#reuse-site-snapshots
4. The script uses identical hyper-parameters utilized in our paper. Simply pass in the site URL (from the CSV) and the snapshot directory. Make sure not to change any of the directory structures or names.
5. We also provide a script that will automatically check the reproducibility. See the instructions in https://github.com/UCI-Networking-Group/AutoFR/tree/artifact-review#reuse-site-snapshots confirm_reproducibility script part.

Results: 1. Follow the below:

2. The terminal will display the rules that are outputted.
3. Open up our "Top5K_rules.csv" (Sec. A.2.2) and look for the corresponding row that matches the zip file name. Then compare the filter rules generated *vs.* the row information. They should match.

A.5 Notes on Reusability

By leveraging the site snapshots we collected in Sec. A.2.2, users and researchers can explore other ways to generate filter rules. This includes:

1. Be less conservative when dealing with site dynamics. For any given site, there are dynamics upon different visits to it. For instance, other images, text, and ads can be served to the same user. We capture these dynamics in our site snapshots by collecting multiple snapshots per-site. Our algorithm randomly selects one site snapshot to test a rule at a given time t step of our algorithm and puts the rule to "sleep" (*i.e.,* remove it from contention) if it does not block any requests. Instead, one can modify the algorithm so that it selects only site snapshots that will cause the rule to block at least one request. We discuss this in Sec. 5.2.2.

2. Explore other RL algorithms. In our paper, we formulate the problem of filter rule generation as a multi-arm bandits problem. Future work can freely explore different RL algorithms offline using site snapshots.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.