# ACCESSIBLE VISUALIZATION PLATFORM FOR ADVERSARIAL NATURAL LANGUAGE PROCESSING

A Research Paper submitted to the Department of Computer Science
In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science in Computer Science

By

Soukarya Ghosh

May 5, 2021

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

ADVISOR
Yanjun Qi, Department of Computer Science

# TEXTATTACK WEB DEMO: WEB PLATFORM FOR EDUCATION NATURAL LANGUAGE PROCESSING

TextAttack is a Python framework for adversarial attacks, data augmentation, and adversarial training in natural language processing (Morris, Lifland, Yoo, Grigsby, Jin, Qi, 2020, p.1). The extensive library, developed by Jack Morris, a machine learning researcher at the University of Virginia, and managed by Dr. Yanjun Qi, is the state of the art in its field, housing over sixteen adversarial attack implementations from published and accredited literature. The ultimate goal of the project is to provide a tool to study adversarial attacks in order to improve existing model performance and robustness (Morris, Lifland, Yoo, Grigsby, Jin, Qi, 2020, p.2). Prior to the development of the TextAttack Web Demo, in order for fellow academics and machine learning practitioners to access these powerful tools, they were required to install the memory heavy package locally and run costly computation on their graphics processing unit (GPU) in order to use the services. In worst case scenarios, where the user did not have a GPU to allow such intensive calculations to run, the process resorted to using cloud computation which took significantly longer and was not consistent in its performance.

In order to alleviate these issues and increase usability of the service by both data science veterans and newcomers, a new method of accessibility was to be introduced. The objective of this technical research is to leverage the industry full stack experience I have gathered in order to build a scalable web application with an accessible interface to allow the TextAttack library to be both demoed and, more ideally, be provided as a service for data scientists. The web application, in its final implementation, should enable users to run all attacks on both custom and pre-fetched datasets, perform data augmentation in real time with the web application's interface, and be able to use an application program interface hosted on the web application's endpoint to run various commands that would otherwise require the installation of the command line interface. In other

words, the goal of this project is to perform a full migration of the TextAttack functionalities and services to a URL endpoint for global accessibility.

## THE FIRST ITERATION

The first iteration of the web application was built using Streamlit, which is a Python framework that allows developers to turn data scripts into shareable web apps through automate visualization and architecture scripts that lets developers skip steps of web development and save time ("Streamlit, Now With Sharing", 2021). Furthermore, Streamlit embraces Python scripting, allowing a pseudo backend to function using Python scripts. It scraps the need for frontend development knowledge, as no HTML or CSS knowledge is required in the programming loop, as this is handled by Streamlit. Additionally, Streamlit allows data caching, which takes care of repeated heavy computation. It stores the computation results, hashed into memory on the server, which makes it perfect for machine learning or data science based applications, where heavy computations happen frequently.

However, even with all of these positives, Streamlit takes away agency from the developer and the platform itself. It is a cookie cutter library that makes every major architectural and design decision by itself, which can be to the detriment of the platform since not every platform has the same needs. For TextAttack Web Demo, there was a need for interactivity and customizability on the client side, which Streamlit was not able to provide. Furthermore, most cloud platforms, such as Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure, does not allow Streamlit applications to be hosted out of the box. The only path to making a Streamlit application available to the public is either to host it on Streamlit's cloud directly or by maintaining a personal server farm. This roadblock in conjunction with the lack of

power provided ultimately forced this research project to have to change the stack to allow a more holistic approach to development.

## DEVELOPMENT AND ARCHITECTURE

There is much to be considered when designing the architecture of any full stack service from scratch, such as the language and framework to develop it in, the servers to host it on, and much more. Web applications have evolved to mature solutions with a plethora of frameworks to build the foundations of a project, which provide sophisticated architecture and base code. At the same time, the introduction of multiple frameworks to achieve the same goal has introduced increasing complexity with respect to getting different frameworks to perform as intended in unison (Kersten, Goedicke, 2010). The choices come down to the need to prioritize certain aspects over another. These include metrics such as load speeds, ease of development, language support, hosting capabilities, integration with tech stack native to TextAttack, and much more. Seeing as TextAttack is a framework only supported on Python at the moment, our search for the right web framework narrowed to the single language.

The most popular Python web frameworks include Django, Flask, and Tornado (Guardia, 2016, p.31). These all have their particular strengths and weaknesses. However, when it comes to building a large-scale project, Django is the most comprehensive tool. This is because this framework is based on the model-view-controller (MVC) software design pattern which divides the related program logic into three interconnected, yet clearly separate, elements. This separation and abstraction of elements allow multiple person projects to thrive, as development on a single portion can be asynchronous of another (Qureshi, Sabir, 2014). Additionally, Django allows more autonomy than most other frameworks, allowing developers to make key decisions on everything from code layout to system and data security.

The next big architecture resource to decide is the host of the platform and how to go about maintaining the service and keeping it scalable. With the rise of cloud computing and many large corporations migrating to cloud giants, such as Amazon Web Services (AWS), hosting on the cloud has become more user friendly than ever before. There are various offspring products of AWS that use its cloud computing capabilities to provide hosting services that prioritize ease of deployment and monitoring metrics. A prime example of this is Heroku, a cloud hosting platform that allows developers to deploy Django apps with the click of a single button and minimal setup. Nevertheless, due to the large array of tools AWS provides for scalability, including an intelligent load balancer, this will be the primary option for hosting for this platform. However, with hosting comes monetary cost that scales significantly as usage increases. In order to pay these costs, we will be applying for grants directly from AWS to either cover a portion or the entirety of the cost. As the service is going to be in development for the foreseeable future, large costs are not to be considered at the moment and will be dealt with as the application gets closer to production.

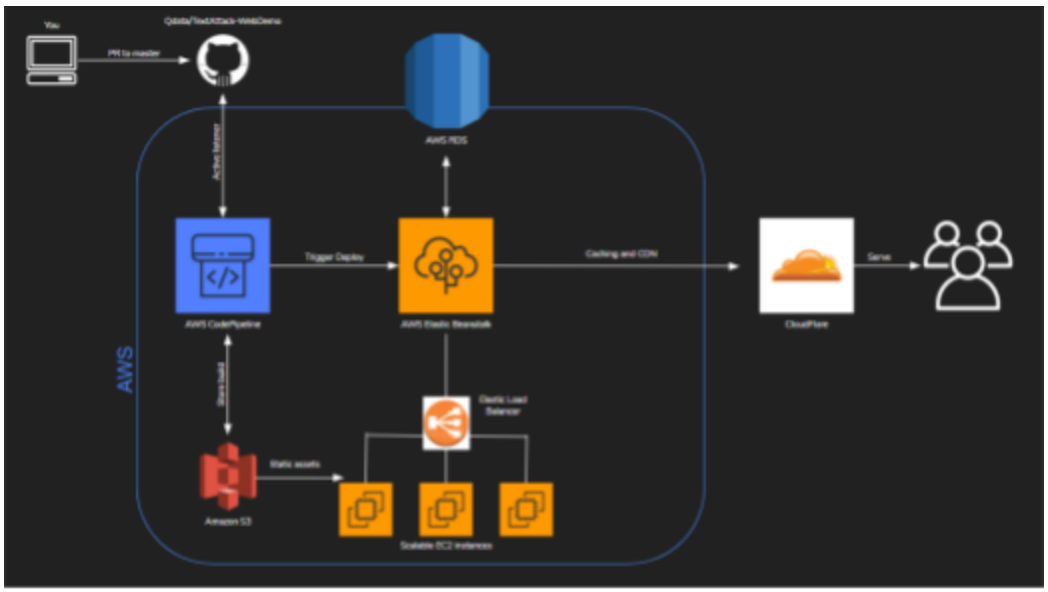


Figure 1. TextAttack Web Application System and Architecture Design. Outlines the flow data from initial user interaction all the way to the resulting interface. (Ghosh, 2020)

The system design shown in Figure 1 details the flow of the code from when it is pushed to the code repository on GitHub, all the way to when the users of the platform get to access said code. The bulk of the diagram focuses on the AWS suite of features, which allows the entire process of deployment, integration, and testing to be seamless and automated. Furthermore, the point of concern of scalability is handled by the Elastic Load Balancer instance in the diagram, which makes sure that users do not face any fault tolerance or single points of failure when trying to access the web endpoint ("Elastic Load Balancing", 2021).

Although this architecture was not fully deployed on the cloud at the termination of the project, this lays out the plans for what is to follow in the future given the continued support of the project by the TextAttack team and subsidiaries. Furthermore, the framework leaves room for introduction for the entire suite of AWS features to be connected, such as AWS machine learning tools that allow NLP algorithms to execute on cloud GPU units for increased efficiency and speed.

**USER INTERFACE OPTIMIZATION**

The entire purpose of this project was to create an intuitive and accessible user interface (UI) for potential clients to test out the library on and get a good understanding of what it is capable of. The first iteration, using Streamlit, gave a barebones UI that had no customization options. The eventual shift to full stack web development technologies removed this hurdle and allowed creativity to flow.

The final version of the UI, as shown in Figure 2,  is minimal, yet effective in its intent. The first view upon accessing the web URL is a form where custom data can be inputted, along with a dropdown specifying a model and algorithm to execute the attack with. Following this intuitive input sequence, commands and functions are automatically parsed and completed in the

backend and a result is returned to the frontend after a few seconds wait. It is important to note that the wait time is heavily dependent on the GPU of the machine executing the backend processes. The eventual deployment to AWS will likely narrow this wait time to milliseconds.

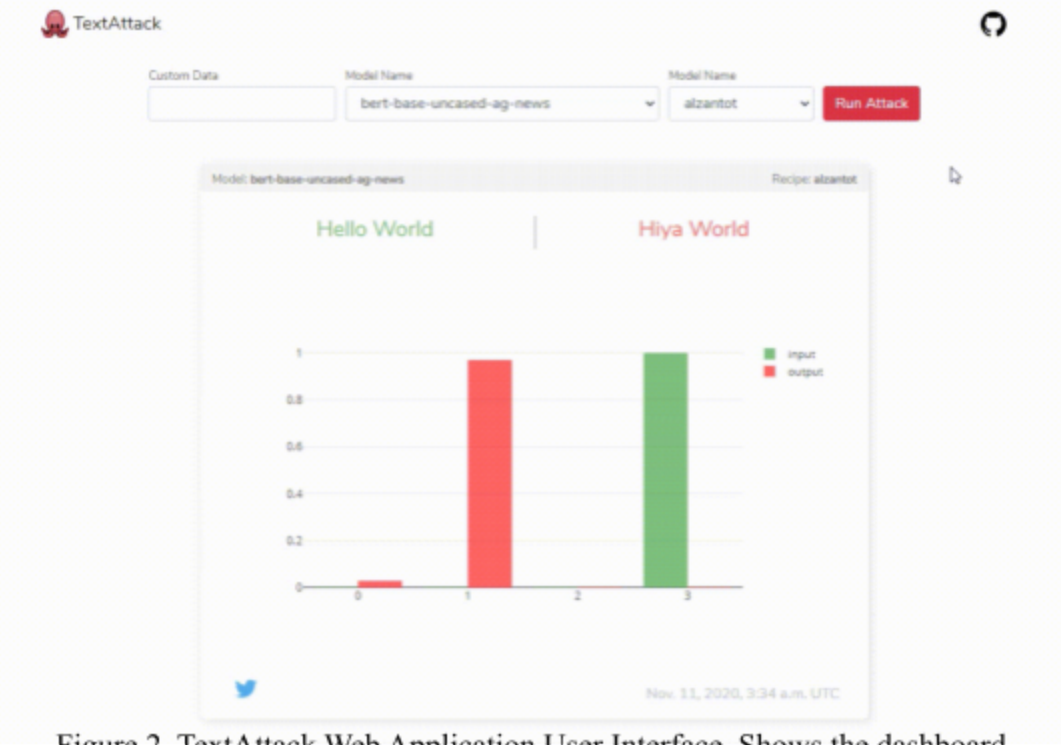


Figure 2. TextAttack Web Application User Interface. Shows the dashboard layout of the main interaction page of the Web Demo. (Ghosh, 2021)

The results are presented in a post format which has some accessibility options as detailed in Figure 1. First, the input test and output text of the attack are shown side by side for ease of comparison, along with a color signifying their classification within the model being used. Furthermore, the confidence of each classification is shown in the histogram to display how much of a difference the attack realistically made to the input text. For comparison and bookkeeping purposes, the model name and attack algorithm are shown on the applicable card, so one can easily go back and see how different attack algorithms manipulate texts differently. Furthermore, there is an ease of access social media button that allows users to share posts to platforms such as Twitter, which ultimately helps increase the TextAttack's visibility.

## FUTURE WORK AND DEPLOYMENT

The initial offering will come with a small suite of features and applications, however, seeing as this will be an ongoing project after my departure, there will be a roadmap of future implementations and paths. Having completed a minimal viable product phase of the web application, I hope the efforts are carried out by future students and members of the TextAttack community. I believe this machine learning framework had potential to grow to a large scale and see greater success and receive more applause than it already has. The web UI built here will be a pathway to increase its visibility to the world and allow users to have easy access to the library's extensive suite of features.

## WORKS CITED


Amazon. (2021). Elastic Load Balancing. https://aws.amazon.com/elasticloadbalancing/.

Chakraborty, A., Alam, M., Dey, V., Chattopadhyay, A., Mukhopadhyay, D. (2018, August). Adversarial Attacks and Defences: A Survey. *Cornell University*. arXiv:1810.00069.

Ghosh, Soukarya. (2020). *TextAttack Web Application System and Architecture Design.* [*Figure 1*]. *Prospectus* (Unpublished undergraduate thesis). School of Engineering and Applied Science, University of Virginia. Charlottesville, VA.

Ghosh, Soukarya. (2021). *TextAttack Web Application User Interface.* [*Figure 2*]. *Technical Report* (Unpublished undergraduate thesis). School of Engineering and Applied Science, University of Virginia. Charlottesville, VA.

Guardia, C. (2016). Python Web Frameworks. *O'Reilly Media*. https://theswissbay.ch/pdf/_to_sort/O%27Reilly/python-web-frameworks.pdf.

Kersten, B., Goedicke, M. (2010, September). Browser-based Analysis of Web Framework Applications. *Cornell University*. arXiv:1009.3714.

Morris, J. X., Lifland, E., Yoo, J. Y., Grigsby, J., Jin D., Qi Y. (2020, May). TextAttack: A Framework for Adversarial Attacks, Data Augmentation, and Adversarial Training in NLP. *Cornell University*. arXiv:2005.05909.

Streamlit. (2021). Streamlit, Now With Sharing. *Streamlit*. doi: https://streamlit.io/sharing.

Qureshi, R. J., Sabir, F. (2014, August). A Comparison of Model View Controller and Model View Presenter. *Cornell University*. arXiv:1408.5786.