Webassembly Video Filters

Team 7

Seth Damany, David Jimenez, Noah Cordova, Hiroya Gojo

Preface

Readme

This is the user manual summarizing the purpose of this project, the features it supports, and how to install or deploy the project. This document will also outline the challenges and novelty of the technologies employed, particularly WebAssembly.

Intended Audience

Video scopes are primarily used in the video production and broadcasting industries to analyze footage and perform color correction/calibration. A browser-based implementation of the scopes provides the most lightweight option for utilizing video scopes, as opposed to physical hardware scopes or costly video editing software.

Related Documentation

- **Halide Language Documentation**: https://halide-lang.org/docs/
- **Webassembly Use without Emscripten**: https://surma.dev/things/c-to-webassembly/
- **WASI Library**: https://github.com/WebAssembly/WASI
- **Video Scopes Technical Guide**: https://www.telestream.net/pdfs/technical/Color-Correction-for-Video-2BW244030.pdf

Product Overview

The goal of this project is to write real-time video scopes capable of running on a browser using Web Assembly code generated using the Halide C++ API. Web assembly and Halide are both technologies that would help increase the performance of video processing, and these combinations of technologies haven't been used before in other projects. Through this project, we explored whether this approach of processing is possible and whether the added technological complexity results in meaningful performance gains.

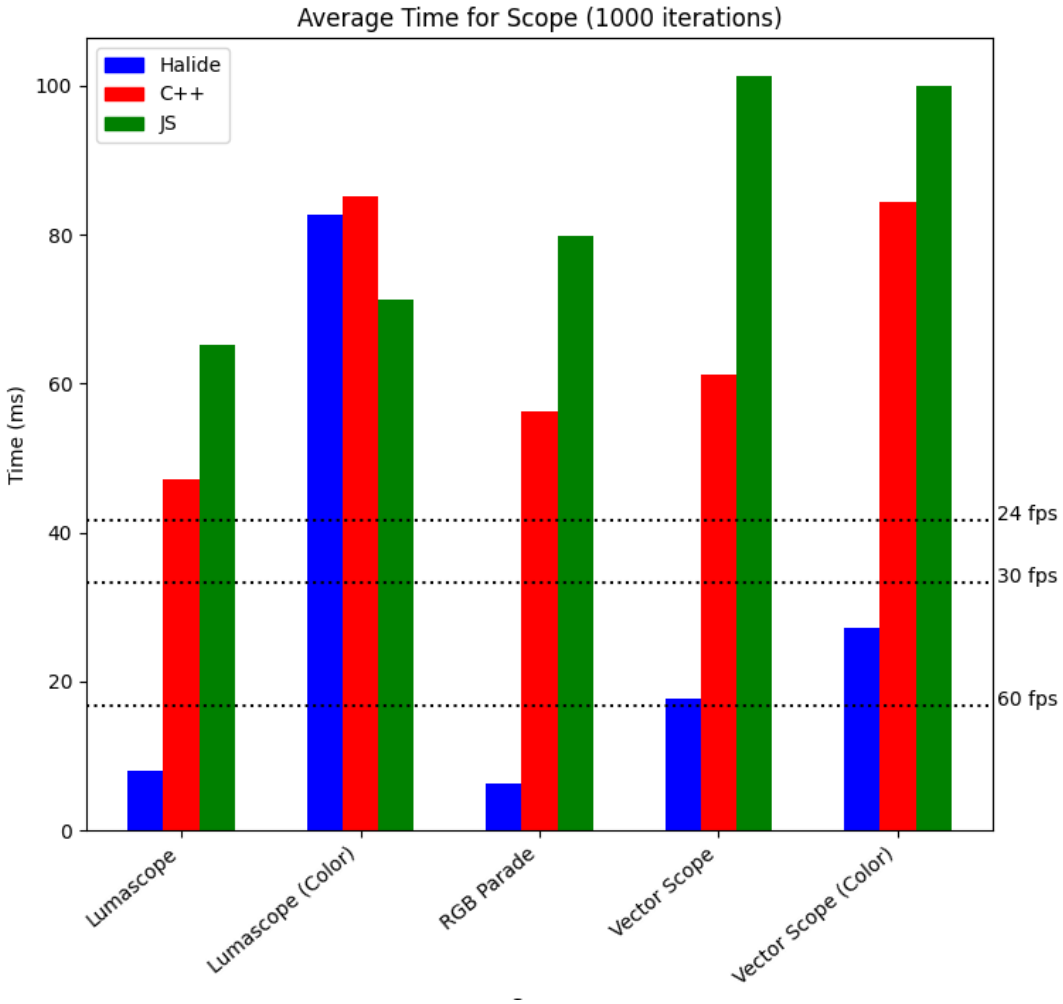
We have implemented the following three primary scopes with Halide/Webassembly, C++/Webassembly, and Javascript. Note that C++ and Javascript scopes were implemented for benchmarking purposes since it allowed us to measure if there were any performance improvements, as well as for development purposes since we would first develop algorithms using C++ then translate it to Halide. However, the overall end goal is to use Halide and web assembly, which would, in theory, result in the best performance results.

- Lumascope (+ color variant)
- RGB Parade
- Vectorscope (+ color variant)

These video scopes all show information about a video frame in different ways, and requires the whole video frame to be analyzed every frame. Hence, performance is critical since many frames have to be processed per second.

Benchmarks

Our findings show that Halide performs as much as 10x faster compared to equivalent C++ and Javascript scope algorithms (see the chart below). However, Lumascope (color) is an outlier in this trend, with both Halide and C++ underperforming to Javascript. We attribute this discrepancy to the scope's algorithm itself, as its excessive memory usage could be further improved upon. Nonetheless, the general performance gains over C++ and Javascript make our work an attractive option for a wide variety of implementations using different targets in Halide compilation, from incorporating the scopes into a fully independent software package, or even compiling an ARM-based version to produce physical scopes.



Note: Benchmarking was done by calculating the average time to process 1000x1000 images over 1000 iterations.

Installation

End User Usage

The website version of the video scopes are hosted on a server at wasm.asucd.dev. End users may visit the site at that URL to access the latest website. Nginx is used as a webserver to host the static front end files, and a node server is running on the backend to send videos on the server to the frontend and allow features such as video uploads. A modern web browser such as Google Chrome and Firefox with the latest updates are recommended since older web browsers may not support web assembly.

The chrome extension can be installed into the browser by going to the chrome extension store and downloading “Webassembly Video Filters.” Note that this is a Google Chrome extension, and while it may work for some other browsers like Microsoft Edge or Chromium, they aren’t explicitly supported and stability can’t be guaranteed on browsers other than Google Chrome.

Then, when the user navigates to a video on Youtube or Vimeo, the scope will show up as a popup on the top left of the page. This can be moved around and manipulated as described in the “Functionalities” section below.

Website Development

For local development, the backend server can be started with node, and the frontend can be started with a python http server or the angular development server. For the first option, there is a script called host.py which creates the python http server which can serve static pre-built content. The latter option is used when the website frontend itself is being developed, in which case there would be many changes in the website so it’s more practical to use the angular development server which tracks changes and automatically reloads the webpage.

For creating a production server, the python http server and angular development server are not sufficient since they are not made to be high performance web servers. Nginx is used instead to serve the prebuilt static files for the frontend, and the backend node server is run by creating a system service on linux. The system service configuration is located in the git repository, as are the nginx configuration file. The production server we deploy our application to is running on a Ubuntu 20.04 Virtual Machine. Finally, the server also utilizes SSL certificates from LetsEncrypt as good practice, which can be setup using a program called certbot.

Chrome Extension Development

The chrome extension can be installed by navigating to chrome://extensions, enabling development mode via toggle, and clicking on “Load Unpacked.” When the file navigator opens up, the user should select the chrome extension folder in the source code.

Whenever changes to the chrome extension source code is made, the chrome extension must be reloaded by going back to chrome://extensions, and clicking on the reload button in the webassembly video filter.

Functionalities

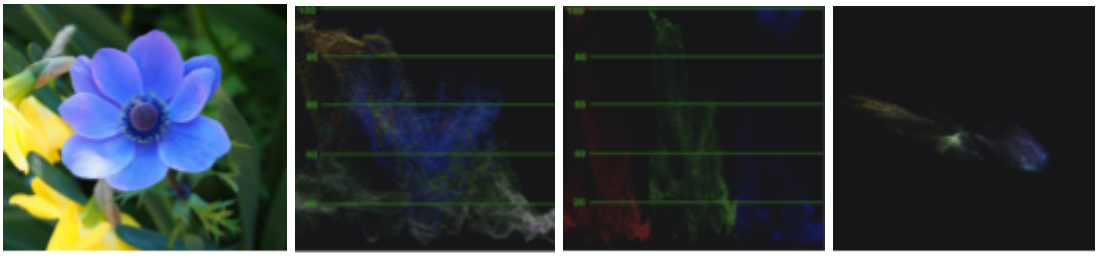
Basic Video Player

The website has a video player which can select between multiple videos on the server, or allow the user to upload a video. This video player has basic play, pause, and volume controls.
The video upload persists until the video is deleted on the server. For the purposes of this project, this was not an issue since this website is mostly to show the web assembly library in use, but in production, this server would need to be modified to either separate videos based on user or delete videos on some regular interval.

Real Time Video Analysis

The website and chrome extension both display video scopes straight in the browser. These video scopes analyze each frame in a video, and display visualizations for the user. These visualizations are updated in real time as the video plays, which can reveal information about a video regarding color, luminance, etc.

The scopes implemented are: **Lumascope**, **RGB Parade**, and **Vectorscope**. Each scope displays different types of information about the video frame. See the images below for reference.



Input *Lumascope* *RGB Parade* *Vectorscope*

Lumascope
Displays the luminance information of the image by column. The color variant (shown above) also displays the pixel's color. If there is an abundance of pixels sharing the same luminance in the same column, the corresponding pixels in the scope will brighten towards white.

RGB Parade

Displays the luminance information for the Red, Green, and Blue channels separately.

Vectorscope

Displays the hue and saturation information. The resulting image is a circular graph where the center signifies zero saturation. The further a pixel is from the center, the more saturated it is in the input. The pixel's angle signifies its hue. The vectorscope color variant is shown above.

Chrome Extension

The chrome extension builds upon the website video scopes and uses the same web assembly file with the same implementation of video scopes.
The chrome extension comes with the feature of being able to analyze videos on other websites such as youtube and vimeo. The extension fetches the video from the HTML page and runs it through the web assembly functions to produce the output image. The output image is then displayed in a small popup which is updated in real time.

The chrome extension comes with other features such as being able to drag the popup around the screen and being able to select between various scopes. There is also an option to disable the popup when the video analysis isn't necessary.

Troubleshooting

General

Halide Installation and Updates

Halide does not have to be directly installed in the developer's system, since the gn build references the Halide library from the git repository under the extern folder. In the case that there is a Halide error or a newer version or Halide is required, the Halide version can be replaced in the extern folder for the given operating system.

Clang/LLVM Version

The most common issue with a first time compilation is making sure your llvm and clang version is up to date. The command `clang --version` should say that your clang version is v11.0+. If not, try installing clang/llvm from a different source.

It is not recommended to compile clang/llvm from source since it'll take several hours. The best way is to find a package manager or repository with the updated version for your operating system.

Memory Issues

When developing algorithms from web assembly, memory issues can be difficult to debug since the browser will suddenly stop running the web assembly coroutine.

In these cases, we didn't use any particular debugger since there is no standardized debugger for web assembly. In general, use of good practice is probably the best bet, usch as making sure to check array overbounds, and double checking that the size of the video scope input and output are correct, taking into account that the images are in RGBA format.

Video Scopes are Outdated

If you make a change to a video scope but the changes aren't reflected in testing, you may have forgotten to rebuild the scope and copy the static library to its appropriate locations, otherwise the browser will still be using the old web assembly static library without the changes. The 3 places where the web assembly file may be outdated are in the chrome extension folder,

webapp folder, and the angular assets folder. It is good practice to replace all 3 of these files after every build.

Windows Specific

Package Manager
Certain package managers presented issues in ensuring that all necessary programs were installed with the latest version and added to the Windows system path. In our experience, the package manager that yielded the best results was Chocolatey.

Mac Specific

Error: LIBCPP_HAS_NO_THREADS cannot be set when STDCPP_THREADS is set

At the moment, this bug isn't fixed in the official WASI repository, so this can be temporarily fixed by commenting out the following lines from:
{project_root}/extern/wasi-sysroot/include/c++/__config

if defined(__STDCPP_THREADS__) && defined(_LIBCPP_HAS_NO_THREADS)
error _LIBCPP_HAS_NO_THREADS cannot be set when __STDCPP_THREADS__ is set.
Endif

Error: Unable to compile to webassembly target.

Mac natively comes with a version of clang. However, by default, this natively installed version of clang and llvm is not compiled with webassembly support, so it is necessary to download clang/llvm with webassembly support.

While there are several ways to do this, the easiest way for Mac is to install clang through brew which installs llvm with webassembly support. The only other issue is that when clang is run from the command line, it must run the brew installed version rather than the native Mac version, so your PATH environment variable must be updated to include the brew version

of clang first before the native Mac version. When you run `which clang` on the terminal, it should show the path to the brew installed clang.

Frequently Asked Questions

Why Halide?

Halide is a programming language embedded in C++ that optimizes performance for image processing. The algorithm and schedule are independent, allowing the programmer to freely and quickly optimize for better performance. Video processing can be an intensive task, so any improvements in performance should be considered and explored.

The primary benefit is not that Halide magically performs computations faster, but that it allows the programmer to quickly switch between different schedules for quick experimentation. Considering the complexity of computers with multi-level caching and complex memory layouts, it is sometimes nearly impossible to reason about the best approach of scheduling, hence it is useful to be able to try different schedules.

Why web assembly?

Web assembly offers a novel way to enhance performance in a browser environment. It's a low level binary instruction format using a stack machine. This enables code to be written for the browser using languages such as C, C++, Rust, and others, which can then be compiled into web assembly and run directly in the browser. For some purposes, Javascript can be too slow since it is an interpreted language and doesn't have the same level of low level control that languages like C/C++ has.

Why videoscopes?

Video scopes are frequently used to gather information from a video frame that may be otherwise difficult to precisely pinpoint. Luminance, hue, and saturation are critical attributes to be considered for color correction during post production. Colorist artists and editors frequently use these scopes as guides to analyze and ultimately refine a video's overall look (see **User Stories** for examples of use cases). Furthermore, scopes are a clear example of video processing through which the strengths of Halide and Webassembly can truly shine.

Why not use emscripten?

Emscripten is a popular toolchain for web assembly, providing an easy to use compiler, wrapper functions for allocating/deallocating memory, linking C/C++ library functions, and debugging support. However, as fully featured as it is, it is also filled with many features that we do not want. Furthermore, emscripten requires more setup for a developer to get the compiler working on their local system, which we wanted to avoid since there was already a high setup curve for the developers of this project already.

Instead of using the emscripten compiler, we compiled C++ files into web assembly using the clang compiler with the llvm backend directly (the emscripten compiler also uses llvm on the backend). Furthermore, we used WASI to provide the standard library implementations for C++ and memory allocation calls such as malloc. Using WASI has a simple installation feature such the library was simply copied to the git repo itself and there is only one extra file that needs to be copied to the developer's system.

Contact Information

Student Team

Hiroya Gojo - hiroyagojo@gmail.com
Seth Damany - iamsethdamany@gmail.com
Noah Cordova - noahco86@gmail.com
David Jimenez - david.jimx5@gmail.com

Telestream

Brian Thomas - briant@telestream.net
Lukas Rickard - lukasr@telestream.net
Troy Fisher - troyf@telestream.net

Glossary

- **Video scopes** - Allows analysis of an image or video by visualizing certain information about the frame in some way. Useful for professionals dealing with images and videos.
    - **Lumascope** - Video scope that displays the luminance information of the image by column.
    - **RGB Parade** - Video scope that displays the luminance information for the Red, Green, and Blue channels separately.
    - **Vectorscope** - Video scope that displays the hue and saturation information in a circular graph.
- **Halide** - A C++ library specializing in image processing and separates algorithm implementation versus scheduling to maximize performance.
- **Web Assembly** - A binary instruction format for the browser based on a stack machine. Has performance benefits over Javascript.
- **WASI** - WebAssembly system interface; provides C/C++ library implementations for web assembly.

Appendix

Introduction

The goal of this project is to write real-time video scopes capable of running on a browser using WebAssembly code generated using the Halide C++ API.
The novel aspect (innovation) in this project is the use of emerging technologies, WebAssembly and Halide, in pursuit of performance gains for real-time video processing.

Technology Survey

Web front-end:

- Vanilla HTML/CSS/JS:
    - Bare bones languages for creating a website skeleton.
    - Useful for prototyping and quick demos.
- React.js
    - Useful for splitting up a website into modular components.
    - Modern, popular, and well-supported framework
- **Angular.js**
    - More fully featured and opinionated than React
    - Built in services, routing, etc.
    - Modern, popular, and well-supported framework
- Chrome extensions
    - Provides a way for snippets of code to be run on any arbitrary site
    - Supports webassembly
    - Can be used to fetch videos from a website
    - Must be installed as an extension, whereas a regular website doesn't require any installation

Web server:

- **Python**:
    - A simple web server and useful for development purposes.
- Nginx/Apache
    - Highly optimized for serving content.
    - Useful for production environments, not necessarily the best for development
- **Node.js**
    - Server side javascript for backends
    - Javascript has many libraries
    - Not intended to be used as a fully fledged web server
        - Needs a webserver to complement it

Algorithm Implementation:

- Vanilla C++

    - Pros
        - More familiarity since everybody knows C++
        - Can be used to prototype algorithms before writing in Halide
        - There are many libraries due to C++ popularity
    - Cons
        - Not optimized and can easily be abused to write bad code
- **Halide**
    - Pros
        - Easily optimized for SIMD, etc.
        - Easy to experiment with different schedules for the same algorithm.
        - Clients have a pre-existing build framework with Halide for us.
    - Cons
        - Unfamiliarity with the language and the compiler

Browser Web Language

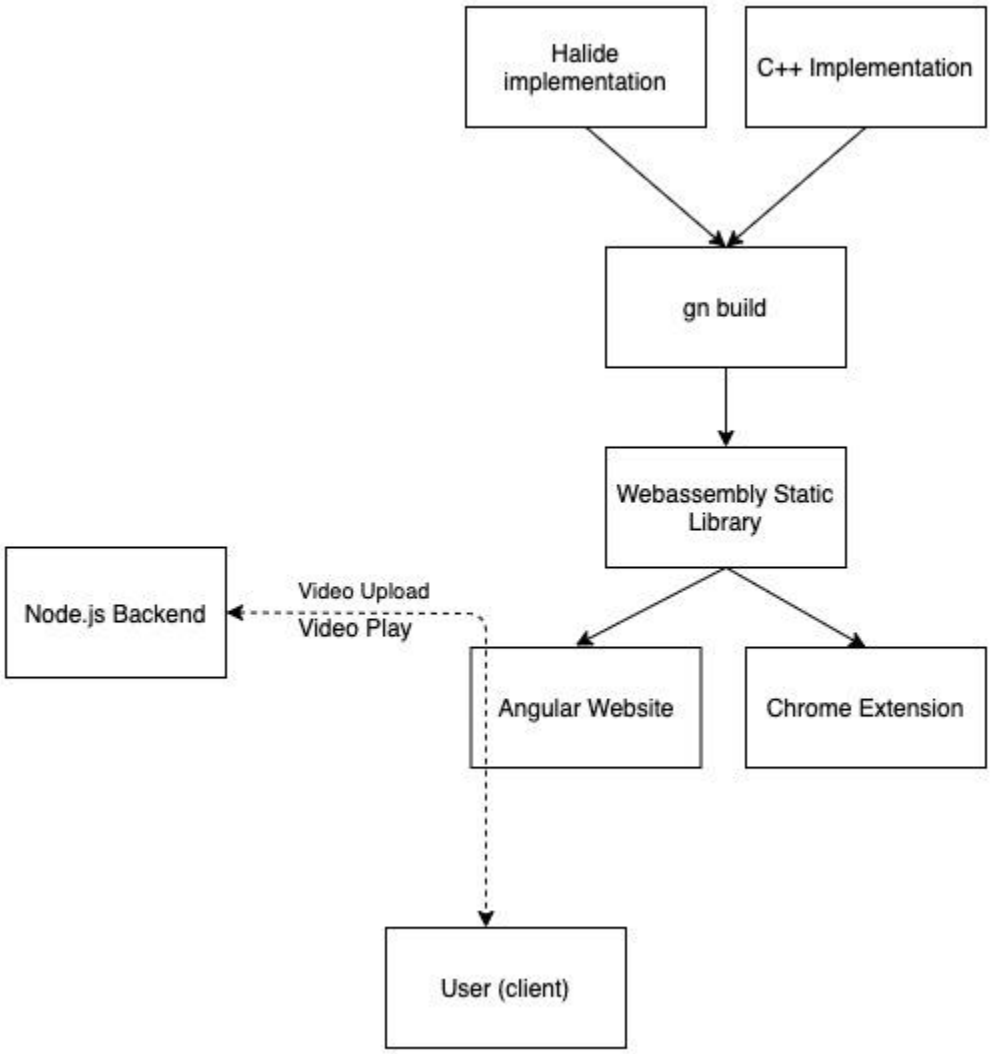
- **Webassembly**
    - Pros
        - Able to port over C/C++ native libraries
        - Novel way to get high performance in a browser environment
    - Cons
        - Difficult to debug
        - Build pipeline is more complex
        - There is overhead from switching to a webassembly function
        - There is overhead from managing memory
- **Javascript**
    - Pros
        - Widely used and well documented
        - De facto standard for web programming
        - Human-readable code
    - Cons
        - Slower than webassembly, making it suboptimal for video processing

Meta build language

- **GN**
    - Enables cross platform builds
    - Generates make or ninja files that are optimized for build time
    - Powerful and clean/readable language
    - Useful since client has a build framework for Halide setup with gn already
- CMake
    - Not as readable
    - Generates make or ninja files that are optimized for build time
    - Widely used for C/C++ projects
    - Enables cross platform builds

System Architecture

A web server hosted on cloud serves the front end website and webassembly files to clients. Backend server with Node.js is used to store and serve videos to the user.



Requirements

1. As a video enthusiast, I want to view color and luminance information about a video on the web as opposed to using a local app or hardware so I can more quickly access it at any time.

2. As a videographer, I want to visualize the color information of my footage so that I can properly set the white balance.

3. As a video enthusiast, I want to use video scopes on popular video streaming sites to get better insight on the hue, saturation, and luminance attributes of a given video.

4. As a broadcast technician, I want to be able to see the color targets so as to ensure the colors in the image are ‘broadcast safe’ and reproducible by most monitors.

5. As a video colorist, I want to see how my footage depicts skin tones so that I know what to adjust during color correction.

6. As a video colorist, I want to see the intensity of color saturation in my video footage so I can accordingly adjust my color filters or neutralize the colors.

7. As a video colorist, I want to visualize the separate R, G, and B channels of my footage to ensure the white balance is consistent.

8. As a videographer, I want to see how bright a given frame of my footage is so I can ensure it is exposed properly.

9. As an archivist, I want to compress video files and compare the video with the original uncompressed video to see if the color composition hasn’t changed too much.

10. As a lighting technician, I want to see the effect light has in different places w.r.t. a camera will change the saturation and luminance intensities of the video output.

11. As a videographer, I want to quickly view which parts of a video frame are overexposed so I can make necessary adjustments.

12. As a video colorist, I want to analyse the scopes of other footage so I can apply a similar visual look to my own footage.

Technologies employed

C++	Primary language for video processing algorithms
Halide	Library used to implement algorithms and schedule them compiler
Clang Compiler	Compiler used in the backend to create webassembly files
GN metabuild system	Generates build files for both the website and the webassembly file

Webassembly	Low level language in browser to enable high performance algorithms.
Python web server	Webserver used to host website static content.
Node.js	Server used to host video files to users.
Angular.js	Frontend framework to develop website.

Cost Analysis

This project is mostly run locally on our own systems and there are no extra costs until we were to host the project on a dedicated server. Since we only need a very simple web server, and not much compute or data resources are necessary, the costs are expected to be minimal.

Social/legal aspect of the product

An ethical concern of this product is that a video must be uploaded to the web server to be analyzed. There is a concern with privacy since the user will trust that we won't abuse their video data. Unlike a local vectorscope or a hardware vectorscope, the web, by its nature, has many more privacy concerns.

The Chrome extension scope-viewer has the additional concern of being used to analyse non-proprietary and potentially trademarked videos. Since our video scopes interpret videos by compressing its chroma and/or luma information, they cannot be used to reproduce the original video. They also cannot stand in for the original video. This helps avoid issues regarding trademarks and copyright.

Our team holds all rights to this product, although the client has expressed some interest in using the final product for their own purposes if it is successful. Our team has decided to release the project as open source with a MIT license so that our project is free to be used and improved upon.