

ABSTRACT

College football is a critical facet of the sports industry today, with billions[5] invested and millions of fans across the world, producing thousands of data points per week. Though many might think that this data is unimportant, it serves a useful role in determining player/team output performance on a weekly/seasonal basis. In this report, we first explore the college football data scene and introduce our platform, Gatornetics, and discuss how we innovate on existing frameworks/arts. After we've given meaning to Gatornetics and discussed the history behind college football data, we then begin to describe the full front/back end build of our web-based platform. Specifically, we discuss the pre-processing stage in the backend, the ML build/API connection, SQL queries for database connection, and react components that all work synchronously together to make Gatornetics the web-based platform it is today. Additionally, once we've discussed current/planned developments, we'll analyze the social impact that Gatornetics has and what our results mean for coaches, data scientists, fans, and the general populace. Overall, we provide a full summary of Gatornetics, describing critical components in each section and what our timeline will look like for the rest of the semester (while also laying out a future development plan for engineers wishing to expand the Gatornetics platform).

INTRODUCTION

Although it may seem that college football is simply a popular athletic attraction for students/fans to attend, it serves as a multi-billion dollar[5] entertainment industry with millions of fans across the world. Given the scale, one is led to believe that this industry produces massive amounts of data, but how is that data presented to fans, data scientists, coaches, and the entire populace? In short, it's 'vomited' out in such a way that is both intimidating and confusing to the normal fan, thus wasting a critical opportunity for people to understand a multi-billion dollar industry. As a result, Gatornetics serves as an alternative to large-scale data-distributors like ESPN, 247 Sports and others, visualizing data in a simplistic manner and offering machine-learning demos that showcase the potential use-cases with said data. In presenting college football data (on a large scale) to the public, not only are we able to shorten the 'knowledge' gap between coaches/analysts/coordinators and fans, but we're able to demonstrate the powerful connection between sports and computing.

BACKGROUND

In regard to previous art/works, Gatornetics was inspired mainly by two different facets that revolved around college football data. The first was ESPN's website, in which they are able to 'vomit' data on thousands of college football players and teams. On ESPN, they not only have all types of data on college football, but a variety of sports as well, thus showcasing their power as a sports-data conglomerate. The other inspiration was from WhatIfSports, specifically their seemingly proprietary game-simulation that allows a user to select two teams from any year and have a random generator (with some internal metrics that account for team-strength-favoritism) produce a score between the two. Now that we've identified the two main platforms, let's begin dissecting each resource and how Gatornetics differs/improves upon each.

As discussed previously, ESPN provides a wide variety of different statistics ranging from individual player contribution to overall team performance metrics (both served on a season/per game basis). However, unless you have years of coaching experience or time spent as an analytics coordinator, it's likely that most of the statistics presented on the website will appear complex and often uncorrelated. This is where Gatornetics serves as a valuable alternative. By visualizing player/team data with graphs/charts on a large scale, we're able to provide the public with a means of analyzing any player/team, given that they have data populated by ESPN (this, of course, addresses the issue with some skill players like O-linemen and D-linemen often having little individual data shown publicly). Although Gatornetics doesn't contain all the specific data that is captured by ESPN, it's able to capitalize on its goal of educating the public and providing correlations between player/team performance in a more refined, simplistic manner.

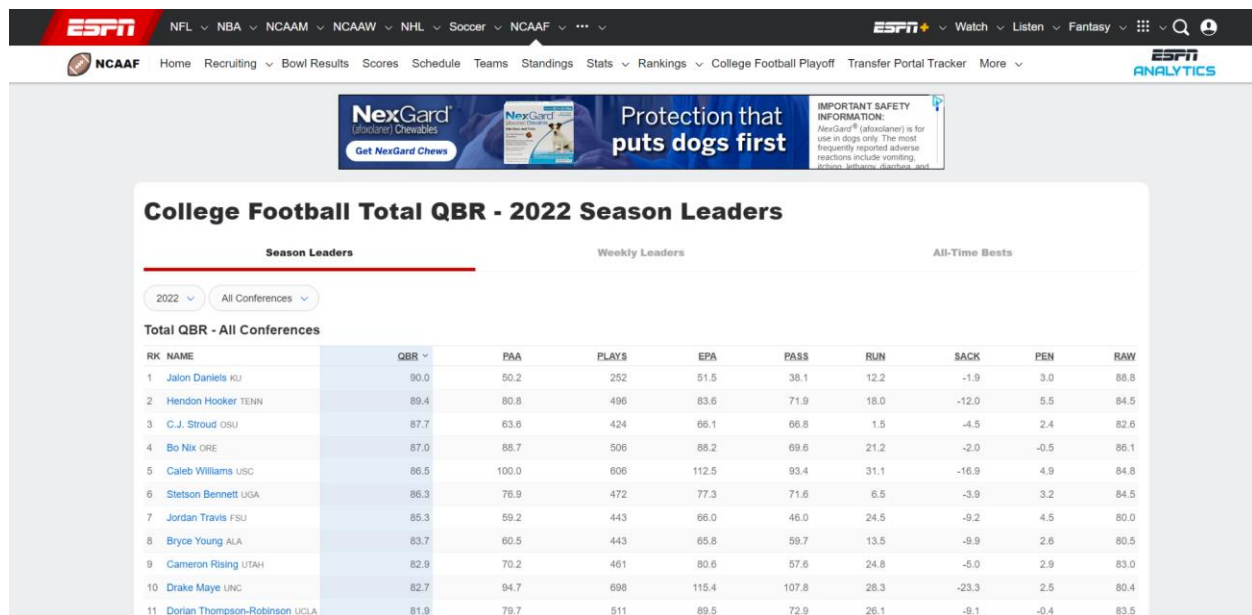


Figure 1: Pictured above is the ESPN Data Vomit Example[1].


Since Gatornetics is mainly presented in two different forms, machine learning demos and large-scale data visualization, it'd be unfair if we didn't discuss one resource that inspired our final-play simulator. WhatIfSports, an web-based platform, provides users with a full game simulation that puts any two FBS teams against one another and generates a score between the two on a quarter-by-quarter basis. Finding information on the architecture behind WhatIfSport's football simulator was rather difficult as it lists all its simulators as 'proprietary work' in their terms of use. However, when one adjusts the parameters to the simulation, we can see drastic differences in the final score, thus implying the usage of a random number generator that strongly influences the results. For example, if we take two top-ten-ranked teams from the 2022 season and have them play in 60 degree weather with wind (as shown below), then the home team wins most of the time, regardless of the other team's strength (in this case, the away team is the winner of the national championship).

2022 Georgia Bulldogs at 2022 Alabama Crimson Tide

Final - 3/22/2023

	1st	2nd	3rd	4th	Total
2022 Georgia Bulldogs	0	3	3	7	13
2022 Alabama Crimson Tide	10	3	0	13	26

Play-by-play:



Think you can run a college football program? Here's your big chance! Get your foot in the door at a DIII school, then prove yourself and you can land that DI dream job. Compete for prizes against your friends in recruiting battles, conference rivalries and more. Set your practice plans, game plans and depth charts. Decide who gets a scholarship and who needs study time. It's all on your shoulders!

Begin your college football coaching career in [Gridiron Dynasty](#).

Scoring Summary

1st Quarter				
BAMA	12:45	TD	Jahmyr Gibbs 4 yd. run (Reichard kick)	0-7
BAMA	4:54	FG	Will Reichard 36 yds.	0-10
2nd Quarter				
BAMA	6:19	FG	Will Reichard 20 yds.	0-13
UGA	1:54	FG	Jack Podlesny 48 yds.	3-13
3rd Quarter				
UGA	12:41	FG	Jack Podlesny 40 yds.	6-13
4th Quarter				
BAMA	13:23	FG	Will Reichard 20 yds.	6-16
BAMA	7:52	TD	Ja'Corey Brooks 16 yd. pass from Young (Reichard kick)	6-23
BAMA	5:00	FG	Will Reichard 34 yds.	6-26
UGA	1:06	TD	Daijun Edwards 1 yd. run (Podlesny kick)	13-26

Team Statistics

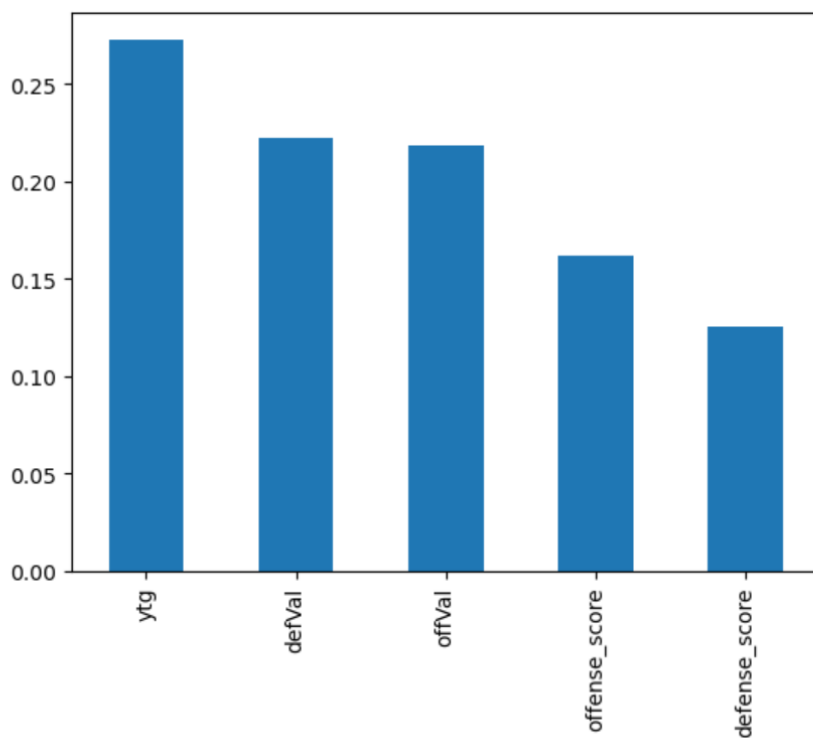
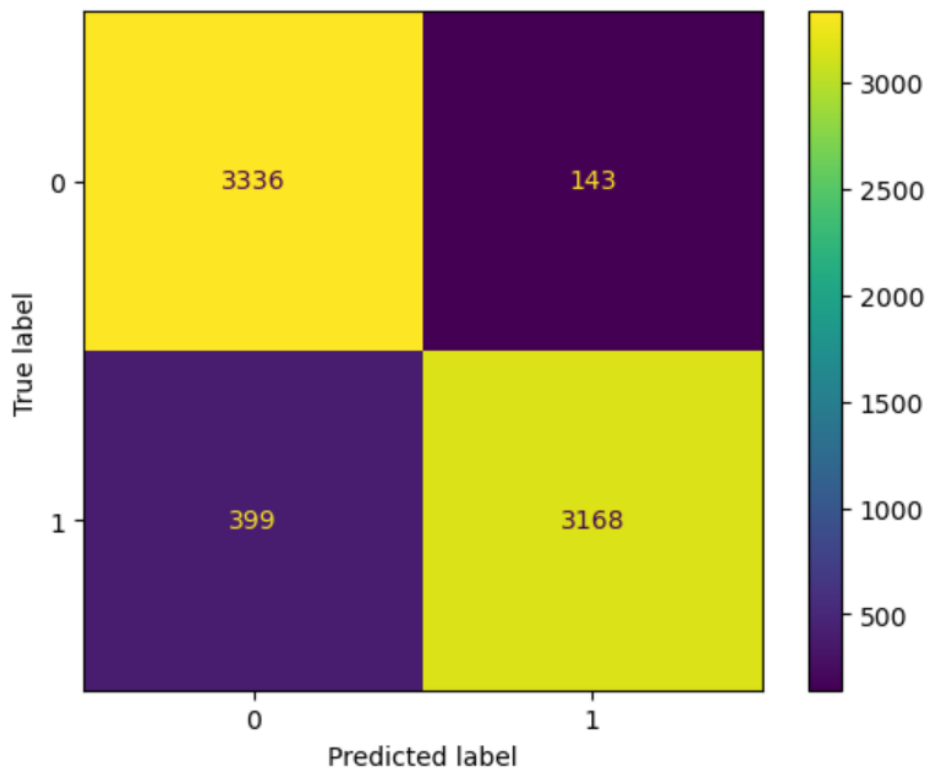
	2022 Georgia Bulldogs	2022 Alabama Crimson Tide
First Downs	17	24
- Rushing	5	6
- Passing	12	18
- Penalty	0	0
3rd Down Eff	3/15	8/19
4th Down Eff	2/3	1/1
Rushes-Yards	38-76	44-164
Avg Rush	2.0	3.7
Comp-Att-Int	20-33-2	25-37-0
Passing Yards	208	283
Sacks-Yards	2-10	1-6
Fumbles-Lost	0-0	1-1
Punts-Avg	6-40.5	4-39.8
KR-Avg	7-18.4	4-16.5
PR-Avg	4-12.0	6-8.3
Penalties-Yard	2-10	2-15
Time of Possession	26:56	33:04

Figure 2: WhatIf Scenario that's highly variable[7].

Although both Gatornetics and WhatIfSports focus on game simulation/prediction, our model strays away from a complete game simulation and instead, focuses more on a final play-call simulator that allows a user to choose the final play of the game. Additionally, our play-call sim utilizes two machine learning models (random forest classifier and k-means clustering) to accurately generate a play result based on the user input. With this build, we stray away from randomization that can be seen in WhatIfSports and move towards more-realistic output that is based on team data from the 2022 season. Later on in this section, we'll discuss some of the testing results from this model and how it corresponds to our strive for accuracy. Additionally, our build offers users more choice that relate to the final outcome, per the pass vs. rush vs. field goal play selection. Overall, our model sim was inspired by WhatIfSports, but differs in many aspects and offers users a new experience that demonstrates machine learning usage with college football data.

Now that we've touched upon the Gatornetics final play-call sim, let's discuss some of the research papers/articles that helped our team during development. While creating the random forest classifier to determine if a play was going to result in positive or negative/neutral yardage, our team first had to do some research on critical features within football-ML research. Blaikie et. al's research paper on "NFL & NCAA Football Prediction using Artificial Neural Networks"[2] and Charles South/Edward Egros's research paper on "Forecasting college football game outcomes using modern modeling techniques"[6] both proposed the idea

that college football ML models are strongly influenced by both qualitative/quantitative variables and that experimenting with different frameworks like regression/classification can yield significant results. This shared concept would strongly influence our synchronous model creation as we'd have to implement pre-processing techniques accordingly to achieve the correct variables in our dataframe (must have good input data for the model). However, our work in the pre-processing stage would be rewarded with a high accuracy score as seen in the images below.



```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Accuracy: 0.9230769230769231

Figure 3: Pictured above are our accuracy results from our GameSim.

DESIGN

Front End

When first landing on the Gatornetics website, users are greeted by a friendly, appealing interface that shows some statistics and a small explanation of how our site works. The site was made as simple as possible for new and old fans of football, allowing users to easily find and use statistics for different players and teams by showing all the important statistics in the form of graphs. Users are also able to easily navigate the website thanks to a full-text search bar, as well as other quality of life features such as the searchable drop-downs found on player and team pages. Compared to other websites that dump all the information on a page, such as ESPN, Gatornetics is a friendly, useful alternative.

Gatornetics facilitates the data flow from the client by utilizing Vercel, our main server host to serve the html and js to the user and then two additional heroku hosted APIs to serve additional requests and data. The PlanetScale database that we use to generate the pages is served by two python scripts that are run on a weekly basis that feed both ml model analysis, and the base player and team data.

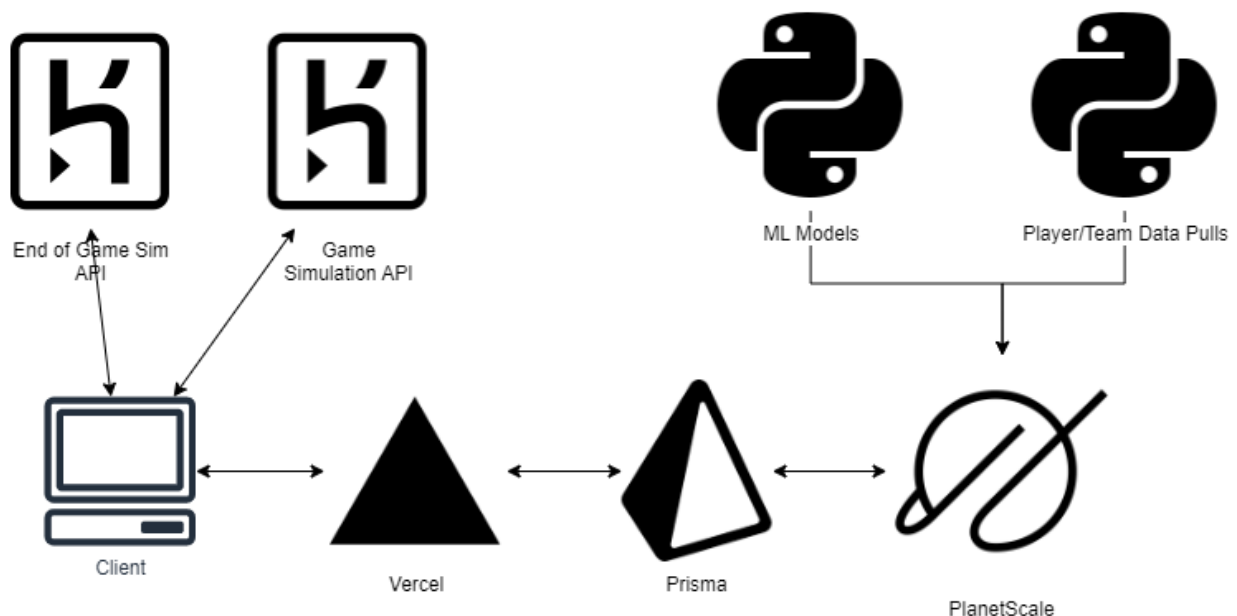


Figure 4: Pictured above is Gatornetics Front End Design.

Back End

Beyond the preprocessing stage and the SQL queries for the planetscale, our entire backend is localized in Python. To grab data, we'll make calls via the cfb api that will allow us to transition onto the preprocessing stage. From there, we'll sort out the data before we package it through successive SQL queries (which pushes the data onto the database (an example of this process can be seen below)).

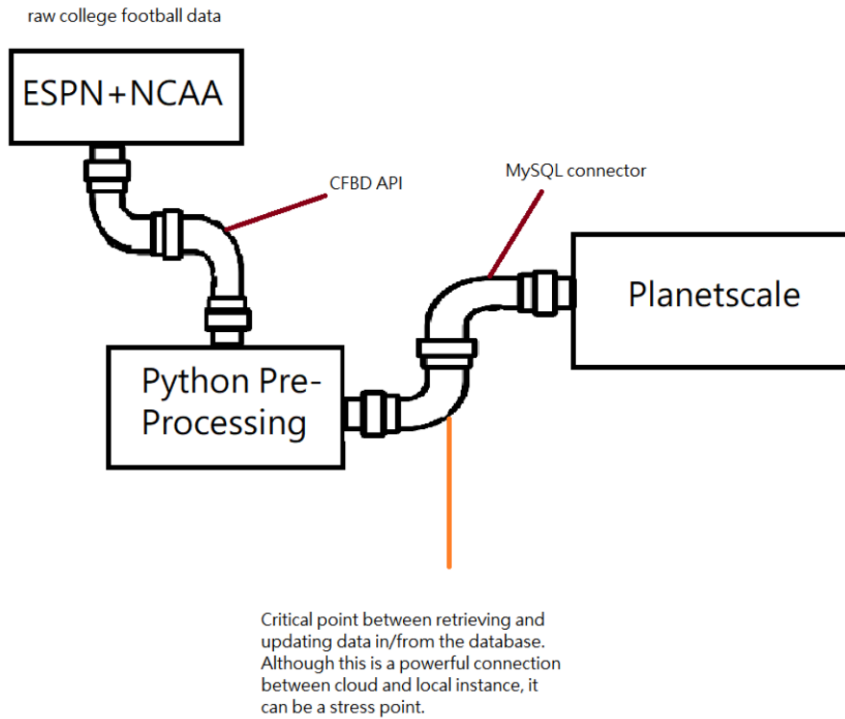


Figure 5: Pictured above is the Backend (data-pipeline) design.

ML

For our game simulator, the model workflow can be seen below. We begin with a random forest classifier that has been trained on all games of the 2022 season for each team. There, we'll pass in data from user input and determine if a play results in positive or negative/neutral yardage. After that classification, we determine the yardage gained based on kmeans clustering models that produce an accurate yardage gain based on the scenario of the play and its best fit on the data from 2022. After that, we produce the yards gained and reveal if the user won or lost based on their input into the model. All models are executed on Heroku and deployed for use on Gatornetics website.

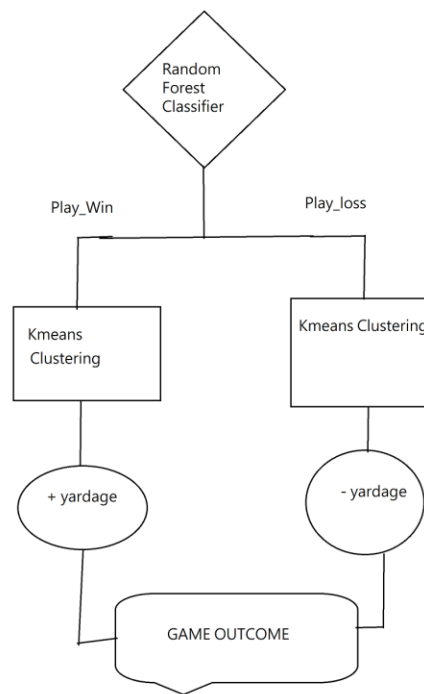


Figure 6: Pictured above is Gatornetics GameSim Model.

Hardware

To optimize our game simulator, we can employ the usage of hardware packages/libraries that allow us to parallelize our code. This process contributes towards the reduction in execution time, which allows the user to not only understand the significance of the hardware/software interaction, but shows them a real-time ML model result in less than 7 seconds. In order to achieve this improvement, we can use Joblibs and specify the number of jobs running on a given CPU to distribute the workload evenly. From this, we can use more of the CPU and achieve a higher runtime. However, we must be careful to optimize using this approach as consistently using all the CPU could lead towards severe errors/charges from Heroku. Shown below is the Joblibs optimization approach towards our ML game sim.

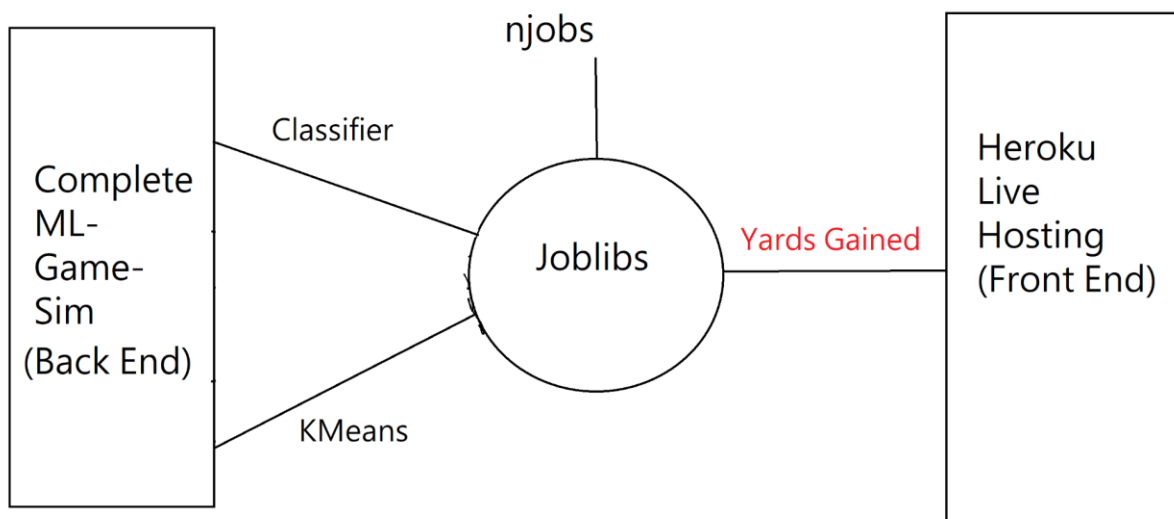


Figure 7: Pictured above is Gatornetics Hardware Parallelization.

IMPACT

We expect our project to impact college football teams which use our product by giving them a statistical advantage using custom built machine learning models. This can aid them in making certain decisions which could be the tipping point in making a loss a win. Ethically speaking, there are no ethical dilemmas or concerns with our project. Teams have their own autonomy in their decisions on if or how to use our product. All the data used by our product is publicly accessible and violates no privacy laws. The cultural impact of our project would be establishing the presence and importance of machine learning in college football which would eventually spread to other sports as well. Economically, this could eventually lead to a market for which, upon success of our product, teams would strive to acquire and develop the best machine learning models and applications for their programs. There are no environmental impacts to consider with this product. The same goes with social impacts as our product is indifferent to race, gender, ethnicity, etc. The benefits of our project have been laid out in its potential to give teams the winning edge. Drawbacks and limitations arise in the reality that predictions are made off a finite number of parameters in a world with infinite parameters and therefore predictions will never be 100% accurate and teams should remain aware of that.

Looking back, it's clear that Gatornetics has delivered on the promises that it first made during the introduction phase. We've demonstrated a clear connection between hardware/software in a field (college football) that has often been thought of as being independent (or without serious technological connections). As stated previously, there are some limitations with Gatornetics, such as the amount of data available or resources we're able to utilize for free. Though you might think that we'd be completely saddled by these restrictions, Gatornetics has challenged this belief and currently stands as a unique platform within the sports data-science world. Overall, Gatornetics has a strong impact within the sports data-science universe as its own, free-to-use standalone platform, available by anyone on the web.

RESULTS

For our alpha milestone Checkly was used for performance and reliability testing. Checkly allowed us to measure the performance of our site in terms of responsiveness and page load times. This is a critical aspect we wanted to test for as it is essential to usability and customer satisfaction. An example of Checkly performance tests on our site can be seen below.

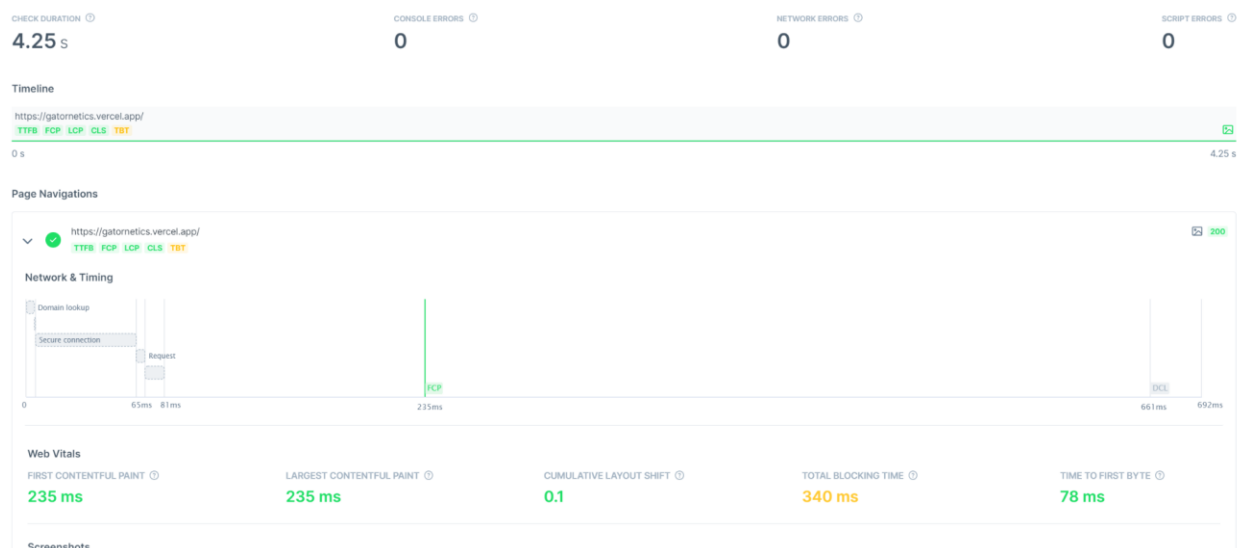


Figure 8: Checkly Results.

Vercel would perform tests on pull requests to ensure that the code sustained a website that could load. It also allowed other members of the team to visualize the changes on a test site deployed by Vercel to see what the changes looked like visually and interactively. A sample Vercel test can be seen below.

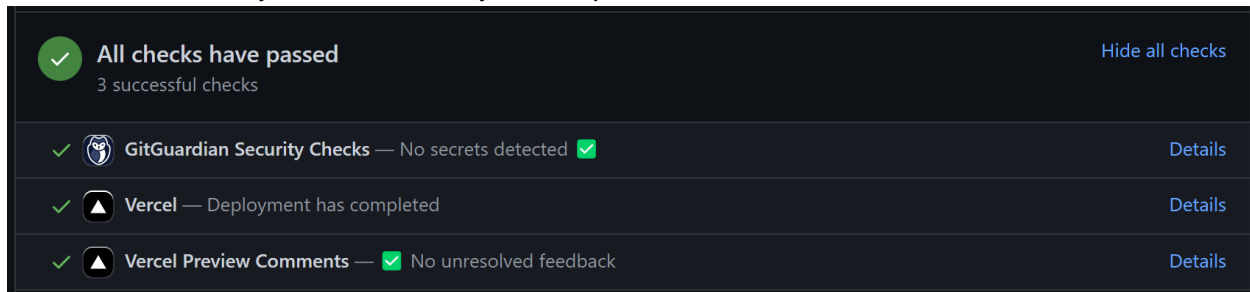


Figure 9: Sample Vercel test.

If any of the above checks failed, Vercel would block deployment to our main branch preventing corrupt code to be pushed into production. We expected our site to pass all of our tests as we had not had an abundance of issues with it so far. Our alpha testing made us comfortable with the level of security of our application and that there are no major security leaks. On the backend side, connection timing tests were used to identify upper bound limits on data pulls.

In terms of the test procedure, in order to stress-test the connection pipeline, we can define a while loop and continuously print the time it takes to retrieve data from Planetscale. In doing this, not only are we identifying the upper-bound on the connection between Python Preprocessing stage and Planetscale, but we are able to determine how much data we can achieve in a single instance. This effectively captures a 'simulation' of what could happen with numerous instances trying to retrieve/update data in the database (shown below are the results from this test).

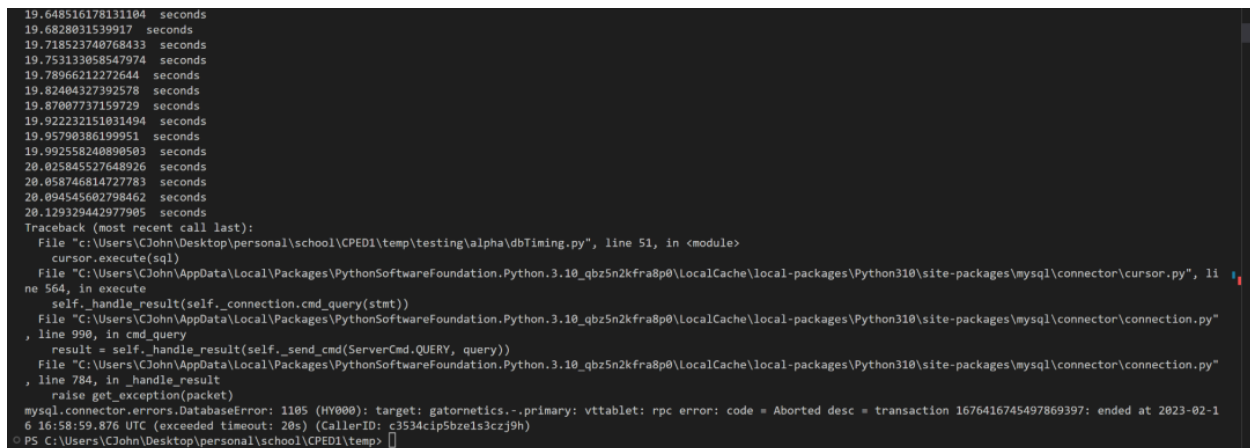


Figure 10: Stress testing database.

The second type of test we can use for the database would be information verification. In our persistent state/internal system, we are pulling data from NCAA/ESPN sources through the CFBD API. In the Python Preprocessing stage, we are performing parsing operations to generate weekly data for each player in FBS (which could be a critical juncture for errors). As a result, our expected behavior is for our data that is being sent to Planetscale to be verified with a source like ESPN through the usage of verification tests.

In terms of the test procedure, now that we've explained the potential for errors, we can begin to describe the process of performing verification testing. Although there exist several solutions for this, we can begin with a small web-scraper from a verified source like NCAA Statistics that will help us verify our information. After we have grabbed the data from an example page on NCAA statistics (Anthony Richardson's stat page for

example), we can begin to write our SQL queries that will retrieve data from Planetscale (specifically the weekly data). During this stage, we should have data from both sources and can thus begin our script that will notify developers of information that is either incorrect/correct. From there, we should be able to automate that test so that It can be applied to all other players. As a result, we are able to verify all our information in Planetscale.

```
Running version: ('8.0.21-Vitess',)
Please Input an FBS team: Florida
CONGRATULATIONS THE INFORMATION IS CORRECT

PS C:\Users\CJohn\Desktop\personal\school\CPED1\temp> & C:/Users/CJohn/AppData/Local/Microsoft/WindowsApps/python3.10.exe c:/Users/CJohn/Desktop/personal/school/CPED1/temp/testing/alpha/verify.py
Running version: ('8.0.21-Vitess',)
Please Input an FBS team: Oregon
CONGRATULATIONS THE INFORMATION IS CORRECT

PS C:\Users\CJohn\Desktop\personal\school\CPED1\temp> & C:/Users/CJohn/AppData/Local/Microsoft/WindowsApps/python3.10.exe c:/Users/CJohn/Desktop/personal/school/CPED1/temp/testing/alpha/verify.py
Running version: ('8.0.21-Vitess',)
Please Input an FBS team: Jaguars
Traceback (most recent call last):
```

Figure 11: Pictured above is database verification testing.

For the machine learning aspect, the expected behavior can be seen below.

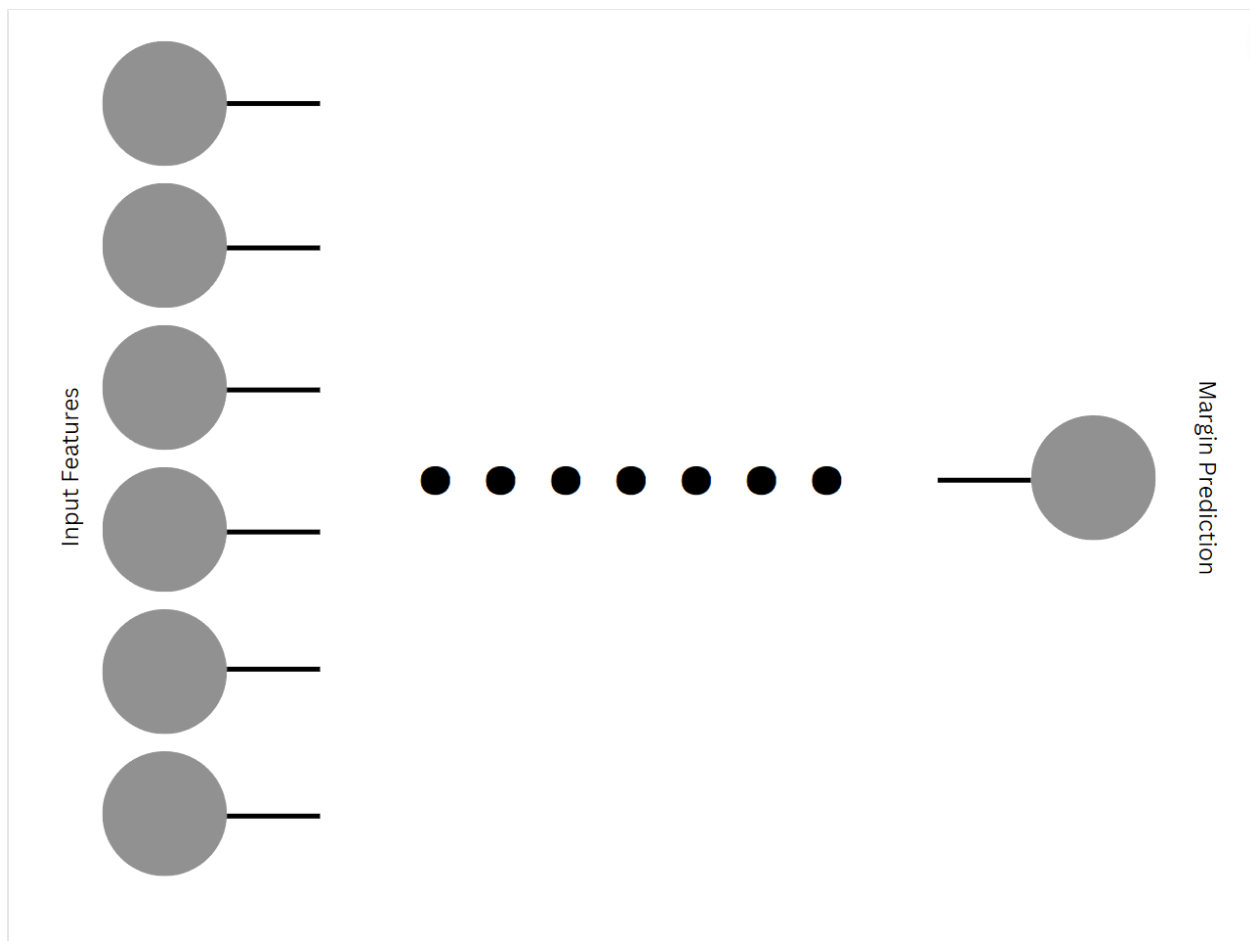
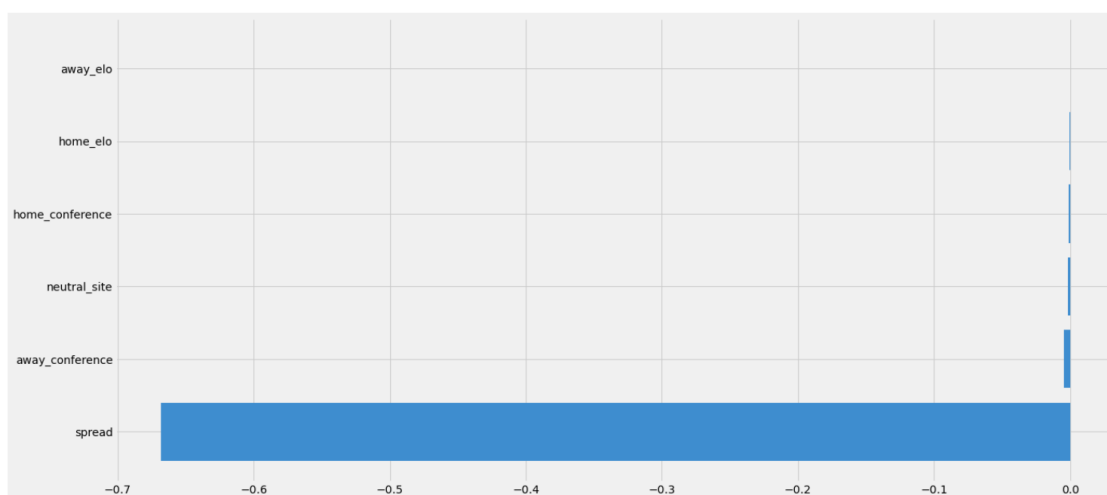


Figure 12: Spread predictor Model.

To put it simply, the expected behavior for the spread ML model is for it to produce an accurate game score margin prediction after receiving a set of input features. The dots between the two outer layers represent the hidden node layers where most of the calculation of weights will take place.

In terms of the test procedures, for the Machine Learning (ML) aspect of Gatornetics, what has been very crucial in creating a model to predict games is choosing the right features to train the model with. Permutation Importance (PI) is the method we have chosen to do this. In essence, PI runs a standard set of data through a fully fleshed out neural network (NN) and analyzes how changing each feature impacts the performance of the NN. Any one feature that has a greater impact on the NN results has greater importance when training the NN. Therefore, it must be chosen. Others that have little to no impact can be omitted as it will likely have no effect or cause unintended effects in training the NN. The features we have chosen with this method are spread, away ELO, and home ELO, away conference, home conference, and neutral site, with ELO being a numeric skill rating that teams hold.



```
In [7]: res.importance
```

```
Out[7]: {'home_conference': -0.0008972482032858454,
         'away_conference': -0.004674110581574709,
         'neutral_site': -0.001802089374190616,
         'home_elo': -0.0007083261475793243,
         'away_elo': 0.00014148480256254386,
         'spread': -0.6680737964281778}
```

Figure 13: Spread feature correlation scores.

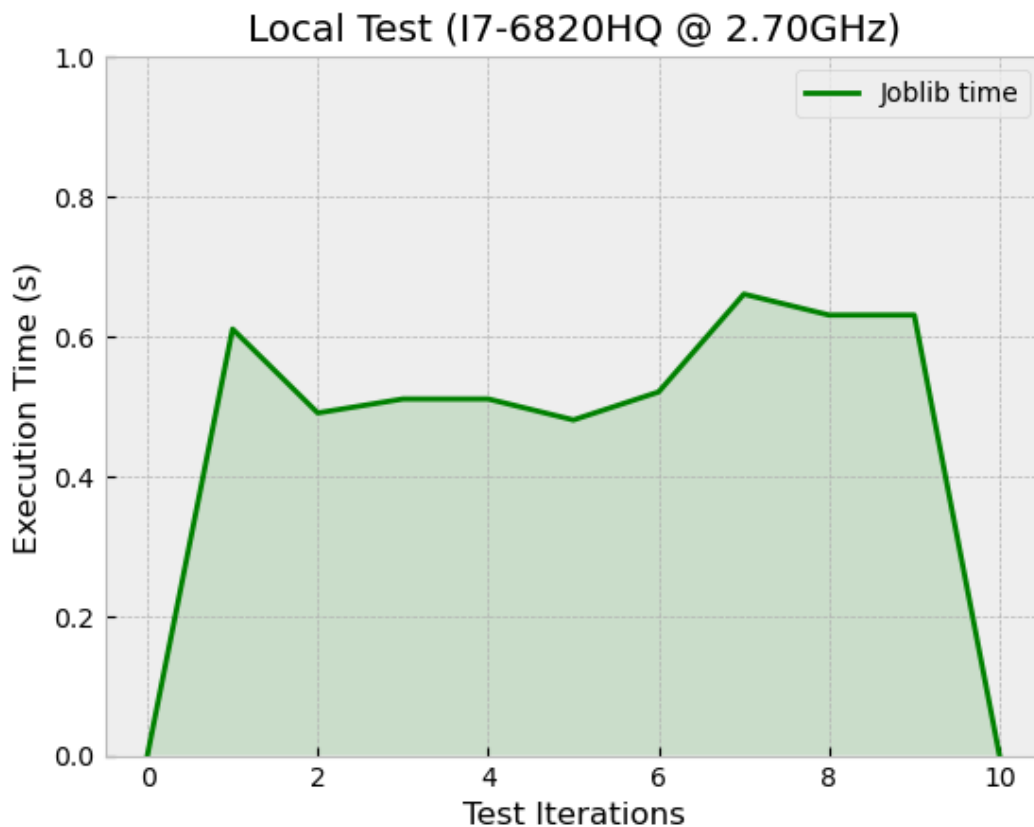
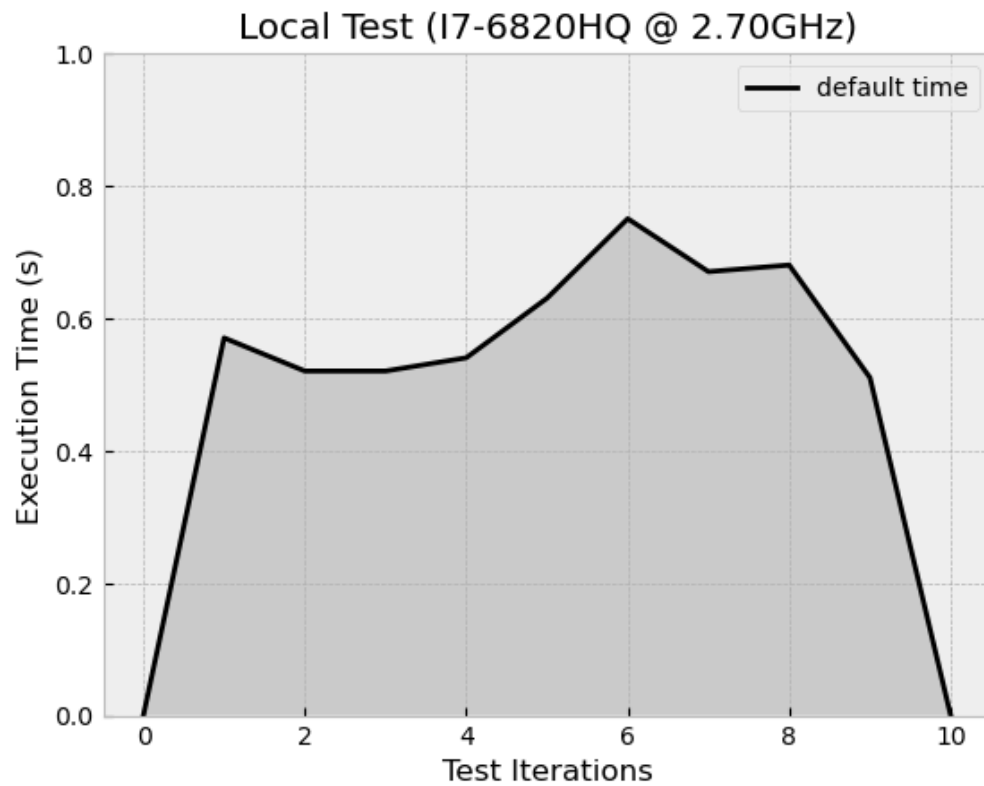
Spread is the main feature that correlates with predicting games correctly with a much greater impact than other features. We must test different combinations of input features, possibly without spread since it is an outlier and may interfere with proper training. It is more likely that spread will have a positive effect, but an exhaustive test would not hurt.

For testing NN quality, we will be measuring training, validation, and testing accuracy to find the optimal point to stop training the model.

We also are utilizing error catching from the api for the spread predictor model to ensure user input matches what we would like it to return with, so in the case of a user inputting a game that does not exist the api will return a 400 error alerting the client to the bad request.

Now that we've defined our machine learning models and discussed their respective tests/results/quirks/etc., we can begin to transition onto the hardware parallelization of Gatornetics. Since we've configured our models and are now leaning towards optimization, we can ask the question: How can we interface with local hardware to parallelize critical functions/actions in our code? To answer this question, we can use certain packages to distribute our workload across a CPU. Although it may seem that we can use any package like Numba that would translate our code into fast machine code with caching (through the LLVM compiler), we must remember that we are using SciKit learn, which will restrict our selection.

After discovering the Numba incompatibility with our Gatornetics ML code, which was a result of Numba's lack of integration with anything other than regular-python or Numpy, we had to experiment with spawning processes and libraries of the like. In our efforts, we mainly focused on the Joblib library and Multiprocessing package, which yielded results that can be seen in the images below:



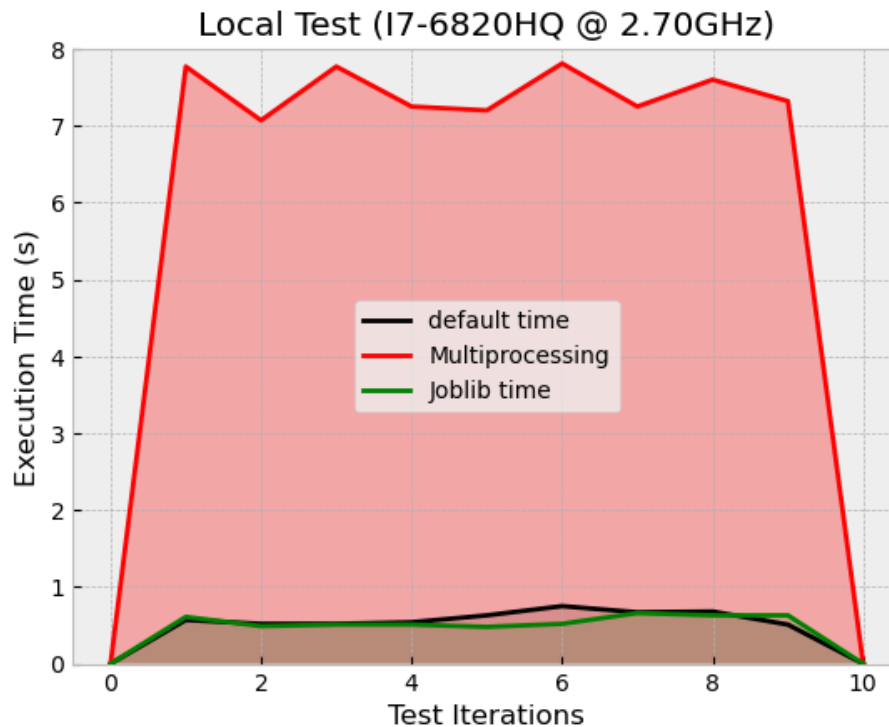


Figure 14: Pictured above are our local test performance graphs.

During our pursuit for answers to our previously discussed question, we decided to split our tests into two categories: Local and Global. Local testing revolved around deploying our models locally and executing various simulations with an I7 CPU. In contrast, global testing involved the usage of an Intel Xeon CPU, which is a result from our Heroku instance (that, subsequently, is hosted through AWS), testing the same simulations done in the local instance. As you can see from the images above, Multiprocessing introduced a significant reduction in performance time, when compared to default/joblib (which both produced comparable timing results). However, once we globally test our optimized/non-optimized model through Heroku, we're able to see a more noticeable difference between our usage/non-usage of Joblib. Specifically, in our later test iterations, we can observe that our Joblib build performs noticeably faster than our default build. This is important as in our testing iterations, the iterations past count 5 are based on unique scenarios from teams that don't normally play one another. While default/joblib may perform comparably in terms of existing/normal scenarios, Joblib's improved performance on new 4th down situations makes the usage of this library worthwhile.

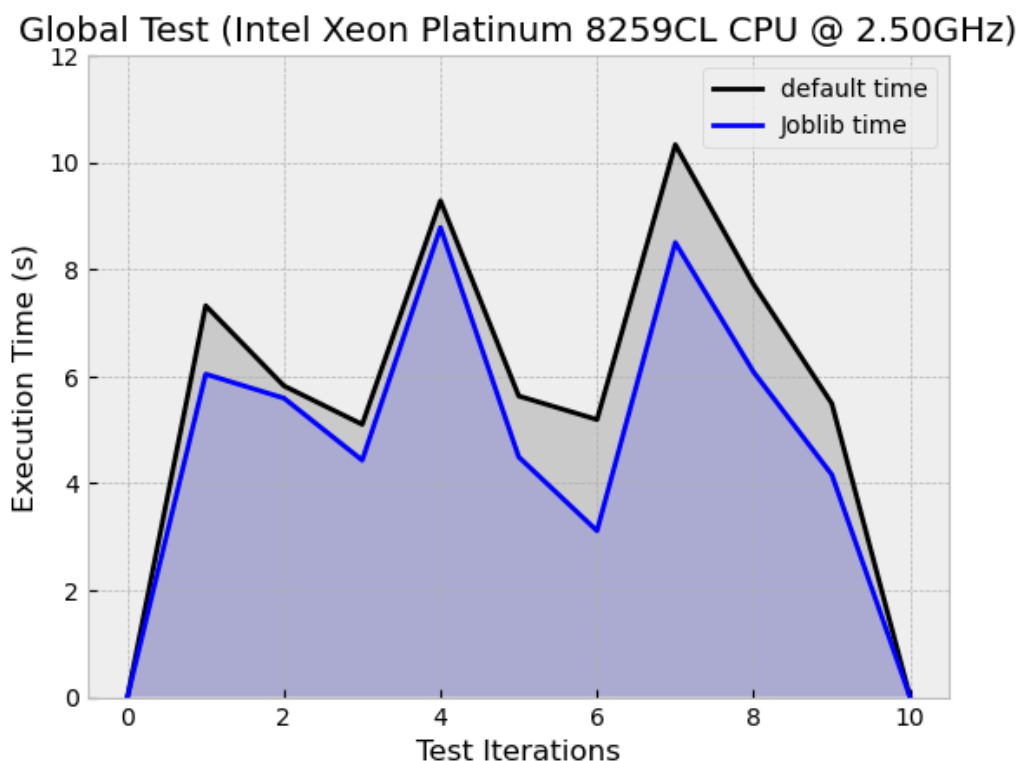


Figure 15: Pictured above is our global test results.

Although we are defining our parallel jobs, we could implicitly call them to use every single core/thread available on our Intel Xeon processor. However, if we were to do this, we would frequently exceed our memory quota on Heroku given our ML model's build and thus change Memory read destinations. This change would result in a significantly slower build, and thus we remain defining our parallel jobs.

Before we transition onto front end testing, let's take a look into the hardware resource % utilization chart as shown below. So far, we've discussed the timing elements of our tests and how they've differed locally/globally. However, we haven't really seen how the CPU is reacting to our multiprocessing packages/libraries. Shown below is a timing diagram taken on an I7-6820 CPU, the same one described/utilized for the earlier timing tests. As we can see from the diagram, when we are specifying our packages/libraries to utilize all the CPU, we can observe a sudden increase in our resource utilization chart. Although this improves our runtime, it can provide to be costly on our local resources and lead to bottleneck issues if allowed to continue over long periods of time.

Previously, we discussed the repercussions of this behavior in regards to our allocation on the Heroku/AWS instance. However, our actions in the local stage can prove to be just as costly as they would through our virtual service. In addition to the bottleneck issue that would increase execution time, one concern we could have would be creating additional heat towards the point of damaging our local hardware. If we were to continuously run our model (through the usage of a while/long-for loop), with 100% of resources being utilized, our CPU would generate an enormous amount of heat in which our system would likely turn off. Due to the potential damage that this instance would create, we (Gatornetics team) are unable to create these conditions and present the results accordingly. However, you can see from the same figure mentioned previously that a specification of maximum parallelization, with the entire CPU being utilized, could lead towards severe problems if left unchecked. Overall, through the usage of Joblibs, we're able to not only visualize our runtime metrics across any given hardware (demonstrating software/hardware connection), but improve ML runtime and thus usability/intractability across the Gatornetics platform.

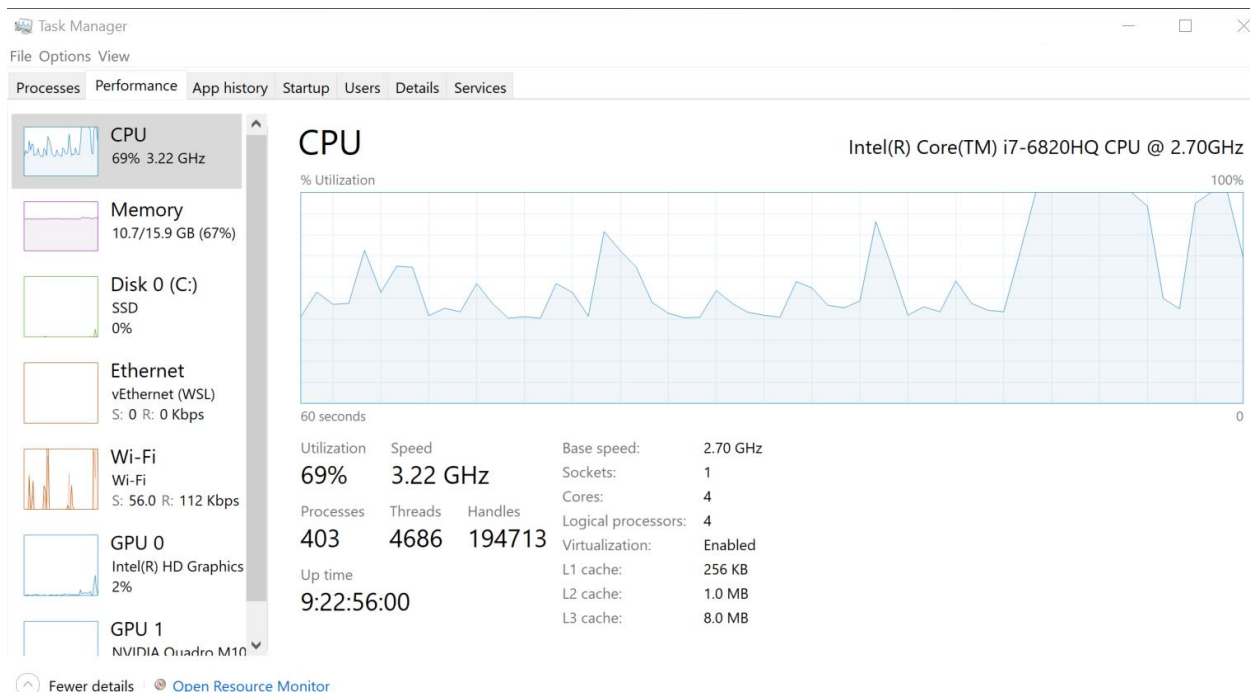


Figure 16: Local CPU % Utilization chart.

For our beta milestone we realized that as more and more data was added, our deploy times began to increase significantly, all the way up to about 8.5 minutes. As a result, we decided to switch from static site generation to server side rendering to increase our sites performance. We again used Checkly to ensure our load times were satisfactory by implementing load time barriers for all metrics. This allowed us to ensure quality user experience on both desktop and mobile versions of our site. In addition, we continued using Vercel to test performance between deployments. At this point we had a well established system for testing and verifying the sturdiness of any code that was attempting to be pushed to production. The flowchart below details a high level overview of that end to end testing system. We felt at this point that our testing was very sturdy and secure but to make our system even more secure we added unit tests which tests our components and functions. These unit tests ensure that components properly render and are displayed with the desired formatting and design. They also test functions to ensure they operate in their intended way and checks their edge cases to ensure there are no holes.

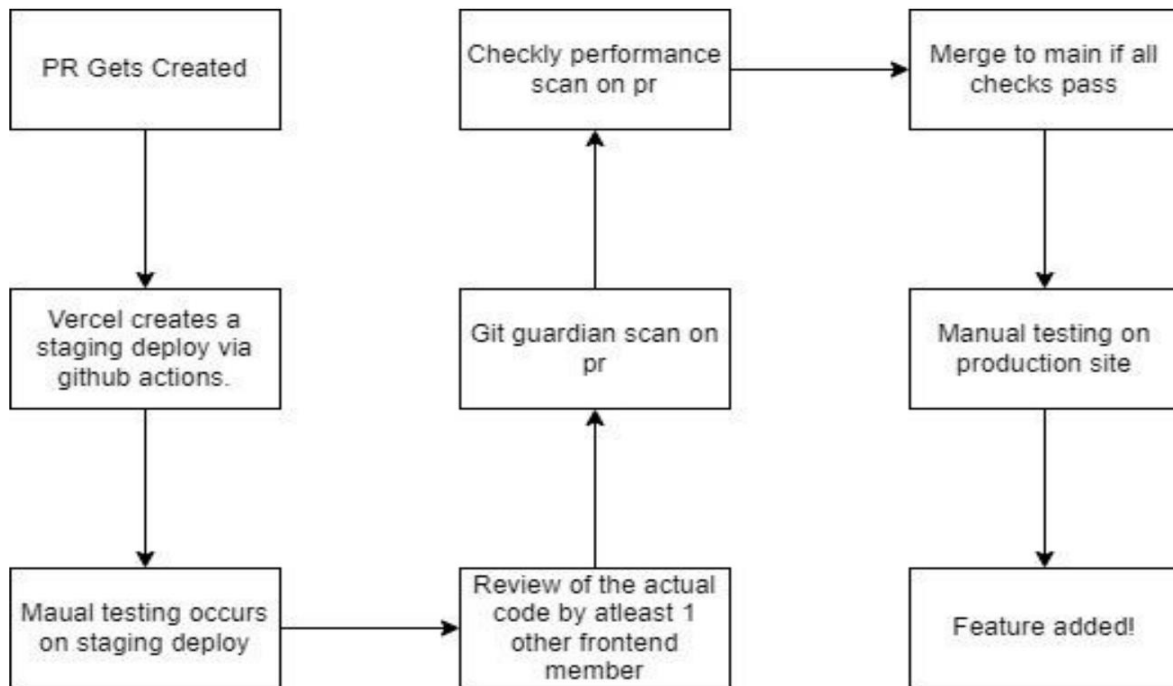


Figure 17: Front End push control.

We also included unit tests as an additional metric to test the functionality of our front end code. These unit tests serve as a way for a developer to check their code before making pull request and having our other, heavier duty, checks run on it such as Vercel checks and GitGuardian. The unit tests can be run and completed in a matter of seconds making it really easy for the developer to see if any fixes are needed. These unit tests were created using the Jest testing framework. They test not only the functionality but also the layout/appearance of our components. Unit tests also allowed us to gauge the code coverage of the tests as seen in the screenshot below. While we did not have 100% of statements covered we were able to get to a point where we felt comfortable enough with the efficacy of the tests.

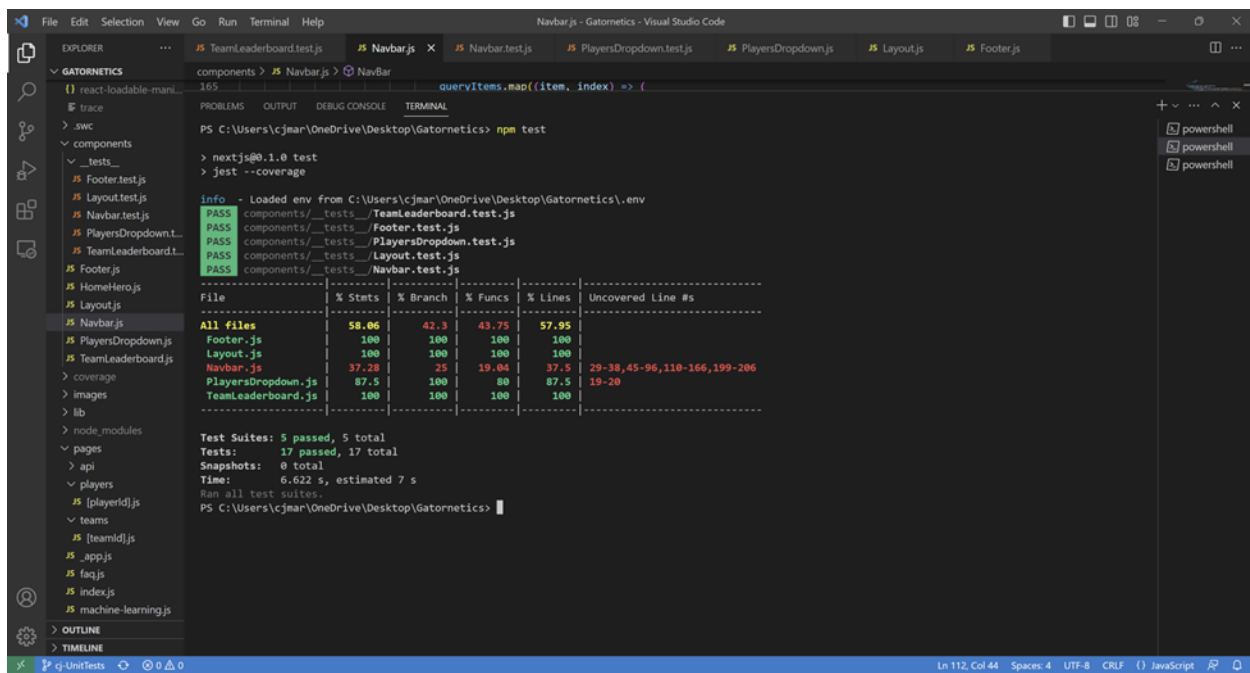
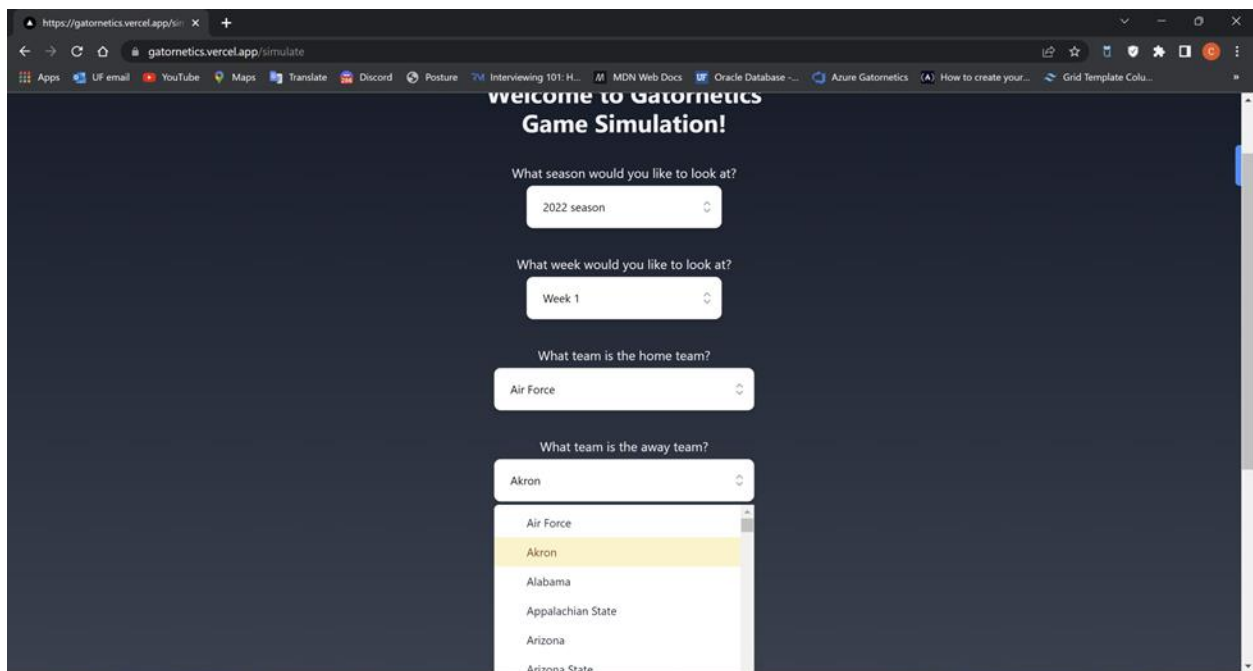


Figure 18: Front End test results.

To run these unit tests, assuming the user has already installed all the necessary dependencies through “npm install”, the process is very simple. To test all files as seen in the screenshot above, the developer should input the command “npm test” into the terminal. As a result, all test files will run and a code coverage statement will be outputted. If a developer only touched one file they can choose to only test that file for a quicker assessment by running “npm test <file name>”. For example, if one only wanted to test the PlayersDropdown.js file, they would run “npm test PlayersDropdown”. This would also output a code coverage assessment for just that file. From these unit tests, we learned that our code was sound and reliable. We did not have trouble getting the tests to pass as a result of weak code. Challenges encountered in the creation and implementation of these unit tests arose from my inexperience with Jest. As a result, there was a bit of a learning curve. The only goal that was not completely finished was to achieve 100% code coverage. However, we felt that the existing coverage was reliable enough and that effort would be better dedicated elsewhere. Some of our front end user interface can be seen in the images below. Overall, our results showcase a fully tested, end-to-end product that incorporates both hardware/software into a portable package for users to utilize in understanding college football on a deeper level.



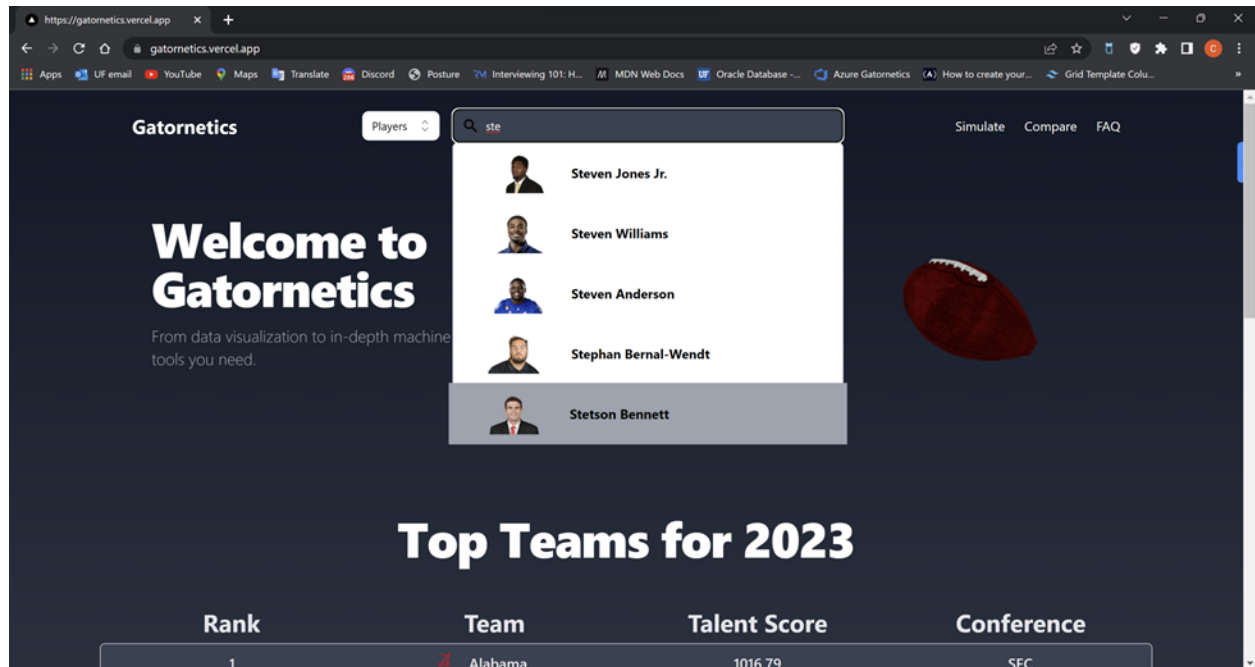
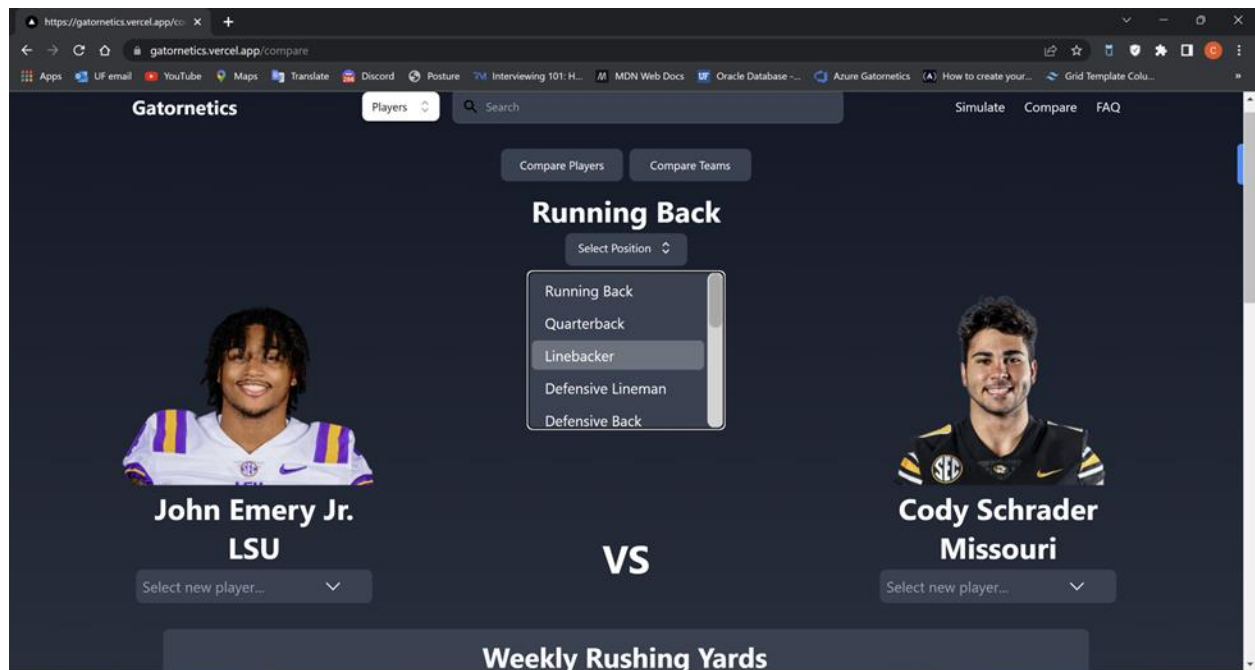


Figure 19: Front End User Interface elements.

Timeline

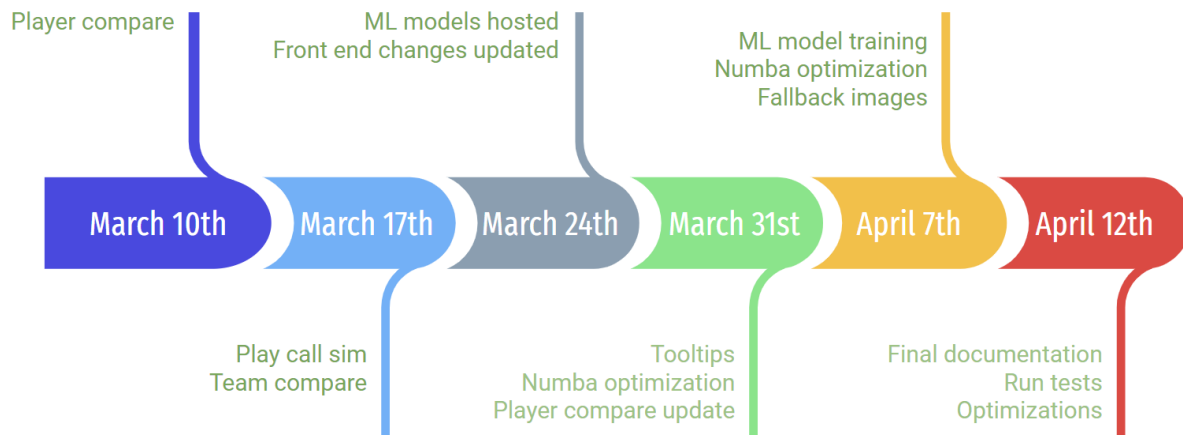


Figure 20: Pictured above is our timeline.

The compare page is complete, providing users the ability to select between comparing players and teams. The newest compare page is more polished and user-friendly than the March 10th/17th page. The machine learning models were hosted around March 24th and have since been trained and incorporated into the website. There were also some visual changes made to the website, such as the new animation on the landing page and the updated leaderboard showing the “Top Teams for 2023”. By March 31st, the tooltips component was created to specify the meaning of certain statistics, clearing any confusion users may have had. As the project nears the end, final fixes were made to ensure that players with no image had a fallback image and that every page runs smoothly and quickly.

CONCLUSION

Planned and designed to be a football analytics platform with unique machine learning models to highlight several arguably important aspects of the game, Gatornetics has turned out to be a great success in the eyes of our team. Generally speaking, our project is as polished as it will get for our two-semester-long development effort. On the front end, all features that were originally planned for Gatornetics have been implemented and refined to look great and work as well as possible. On the backend, the database is fully fleshed out with all the data our team originally planned to store in it. For the machine learning aspect, we have several models that have been trained, validated, and tested to perform as well as possible. To say the least, our project is pretty much finished. Our leftover goals are to keep improving upon the machine learning demos of Gatornetics as our team is continually inspired by new, fresh ideas and further testing to verify that the site is working properly.

REFERENCES

[1] "2022 College Football Season Leaders Total QBR." ESPN, ESPN Internet Ventures, <https://www.espn.com/college-football/qbr>.

[2] Blaikie, Andrew et al. "NFL & NCAA Football Prediction using Artificial Neural Networks." (2011).

[3] Cfbd. "CFBD/CFBD-Python." GitHub, <https://github.com/CFBD/cfbd-python>.

[4] Nflverse. "Nflverse/Nflverse-DATA: Automated Nflverse Data Repository." GitHub, <https://github.com/nflverse/nflverse-data>.

[5] Rittenberg, Adam. "Big Ten Completes 7-Year, \$7 Billion Media Rights Agreement with Fox, CBS, NBC." *ESPN*, ESPN Internet Ventures, 18 Aug. 2022, https://www.espn.com/college-football/story/_/id/34417911/big-ten-completes-7-year-7-billion-media-rights-agreement-fox-cbs-nbc.

[6] South, Charles and Egros, Edward. 'Forecasting College Football Game Outcomes Using Modern Modeling Techniques'. 1 Jan. 2020 : 25 – 33.

[7] "What Is SimMatchup College Football?" SimMatchup College Football - Free NCAA Matchup and College Football Sim Games, <https://www.whatifsports.com/ncaafb/default.asp#top>.