SmoothRide: AI Pothole Detection

Group L30

Team: Nicholas Gray, Andrew Anchieta, Aaron Netterstrom, Thomas Wallsmith, Kyle Williams, Mason Williams

Table of Contents

1 Executive Summary **6**
2 Project Overview **7**
2.1 Project Description 7
2.2 Motivations **10**
2.2a Nicholas Gray 10
2.2b Kyle Williams 11
2.2c Thomas Wallsmith 12
2.2d Mason Williams 14
2.2e Aaron Netterstrom 16
2.2f Andrew Anchieta 17
2.3 Societal Impact **18**
2.3a Disability 18
2.3b Assisting Local Governments 18
2.4 Issues **19**
2.4a Legal 19
2.4b Ethical 22
2.2c Privacy 23
Location Data 23
Camera Access 24
Other Considerations 25
3 Project Characterization **26**
3.1 Goals Overview 26
3.2 User Stories 27
4 Requirements **28**
4.1 General Requirements 28
4.2 AI Requirements 29
4.3 Mobile Requirements 31
4.4 Laptop Program Requirements 32
4.5 Miscellaneous Requirements 34
4.45a General Requirements 34
4.5b Database Requirements 34
4.5d Map Requirements 35
5 Concept of Operations **36**
6 Division of Labor **39**
6.1 Timeline of Roles 39
6.2 Individual Contribution 40
6.2a Nicholas Gray 40

6.2b Kyle Williams 41
6.2c Thomas Wallsmith 42
6.2d Mason Williams 43
6.2e Aaron Netherstrom 44
6.2f Andrew Anchieta 45
7 AI Research 47
7.1 Programming Language 47
7.1a Python 47
7.1b C++ 49
7.1c Julia 50
7.1d Mojo 51
7.2 Deep Learning Library 52
7.2a TensorFlow 52
7.2b PyTorch 53
7.3 Task Selection 54
7.2a Image Classification 54
7.2b Object Detection 55
7.2c Object Segmentation 57
7.4 Model Selection 59
7.4a YOLOv8 60
7.4b YOLOv5 61
7.4c EfficientDet 61
7.4d MobileNetV2 61
8 Data Collection 63
8.1 Data Requirements 63
8.1a Labeled 63
8.1b Clean 64
8.1c No Obstructions 65
8.2 Found Datasets 66
8.2a RDD2022 66
8.2b Road Damage – Dataset Ninja 67
8.2c AI Pothole Detection Computer Vision Project 68
8.2d Road Pothole Images for Pothole detection 69
9 Dataset Hosting 70
9.1 Dataset Divisions 71
9.2 Dataset Uploading 72
10 Model Training 73
10.1 Training Setup 74
10.2 Results 76

11 Mobile Design and User Experience **79**
11.1 Application Pages 79
11.2 Figma 81
11.3 Camera Screen 82
11.4 More Information Screen 84
11.5 App Drawer 85
11.6 Settings Screen 86
11.7 Mapping in Mobile Application 91
11.8 Libraries Used 92
12 React Native vs Flutter **93**
12.1 Flutter, what is it? 93
12.1a Early Development and Origins 94
12.1b Key achievements and enhancements 94
12.1c Recent Developments and Future Prospects 95
12.1d New Material 3 Widgets for Flutter for Web 97
12.1e Drawer for Navigation (Navigation Drawer) 97
12.1f Improved DevTools 97
12.1g Utilizing Dart 99
12.1h Flutter Interact 99
12.2 Why Flutter over React-Native 99
12.2a Widgets and Compatibility 100
12.2b Mobility 100
12.2c Open Sourced 101
12.2d Shorter Time to Market 101
12.2e Unified App UI and Business Logic Across All Platforms 102
12.2f Hot Reloading 102
12.2g Shorter Testing Process 103
13 Mobile Application Architecture and Logic **105**
13.1 Multi-platform vs Native Frameworks 105
13.2 Logic and Architecture 107
13.3 Presentation layer 110
13.4 Domain Layer 113
13.5 Model Integration 116
13.6 File structure 117
14 Laptop Program **118**
14.1 Motivation 118
14.2 Hardware Used 118
14.3 Language and Libraries 119
14.4 Overall System Design 120

14.5 Initial Startup Screen 122
14.6 Video Feed Page 123
14.7 Lane Setting System 124
14.8 Alert System 125
14.9 Pothole and Image Uploading 126
14.10 Settings 128
14.10.1 Detection Settings 128
14.10.2 Alert Settings 129
14.10.3 Video Settings 130
14.11 Mapping Page 131
14.8.1 Pothole Images 132
14.8.2 Bounding Box Marker Selection 132
14.8.3 Adding / Deleting Potholes 133
14.9 Report Generation 133
15 Database and API 134
15.1 Introduction 134
15.2 Database Overview 137
15.3 Supabase 138
15.4 Original Database Plan 142
15.5 Final Database Structure 148
15.6 Final API Design 148
15.7 Old API Design 149
15.8 Conclusions 156
16 Front-End Web Application 157
16.1 Web Application Overview 157
16.2 Web Application Architecture 158
16.3 Web Application Style 158
16.4 Web Application Maps API 159
16.5 Web Application Deployment 161
17 Testing and Evaluation 162
17.1 Model Testing 162
17.2 Application Testing 164
18 Conclusion 167
19 Administration and Operations 169
19.1 Tools 169
19.2 Budget 171
19.3 Timeline 172
19.4 DevOps 174
18.4a Version Control 174

19.5 Agile Format 178
20 Acknowledgements **179**
20.1 References 179
Appendix **190**
Permissions 190
Links 190

1 Executive Summary

The most common mode of transportation in the United States is personal motor vehicles. The average American spends about 55 minutes driving every day driving 29 miles per day (United States Department of Transportation, 2017), totalling up to a yearly average of 20,075 minutes and 10,585 miles per driver and totalling to 3.17 trillion vehicle miles traveled on US roads in 2022 (Carlier, 2023). With this many vehicles and mileage on US roads, road damage is a significant concern for both road and vehicle safety. Around one-third of the approximately 33,000 traffic-related deaths each year involve poor road quality as a factor in the accident (Pothole.info, 2023). Furthermore, AAA found in 2021 that 1 in 10 drivers sustained vehicle damage that required repairs after hitting potholes, totalling to $26.5 billion in damages in 2021 (Edmonds, 2022). With this danger to driver and vehicle safety and lack of repairs, there is a need for some type of assistance to help drivers reduce accidents and damages, and for municipalities to track potholes for further removal.

SmoothRide is an AI-powered portable application to solve this issue easily and simply for drivers and municipalities. The objective of SmoothRide is to provide an AI-powered laptop program to notify drivers of upcoming potholes in their lane, then store the location of the potholes in a database that can be accessed at any time in an easy-to-view map.

The driver will put their camera on the top-right of the windshield, giving a clear view of the road while driving, and connect their camera to our program on a laptop. While on the road, SmoothRide will be running our object detection AI model on the camera to look for any upcoming potholes. If a pothole is detected in the lane that the user sets, the driver will receive an auditory alert, and the pothole will be sent to a database that can be viewed at any time via a live-updating website. The user can manually add potholes found, delete potholes that were falsely detected or are now removed, view an image of the pothole detected, and generate a report of selected potholes’ location.

2 Project Overview

2.1 Project Description

Driving is the primary means of transportation for most individuals in the United States. As reported by the US Department of Transportation, 87% of daily trips are done in private vehicles, and 91% of people commute to work using private vehicles. On average, a single driver in the US spends 55 minutes on the road and drives an average of 29 miles per day, totalling up to about 4 trillion miles driven in the US per year (United States Department of Transportation, 2017). With all this driving, however, there is a concern about road quality and damages on US roads. The US Department of Transportation reports that approximately 80.95% of overall roads in the United States were of acceptable quality in 2020, with a range of 94.60% in Tennessee and 8.61% in Washington, DC (United States Department of Transportation, 2020). However, in major cities, the quality is much smaller, ranging from 64% in Los Angeles, California to 38% in Hartford, Connecticut (Pothole.info, 2023). Given that approximately 80% of the US population lives in urban areas, this lack of road quality represents a possible danger to driver and vehicle safety (United States Census Bureau, 2023). There is a present danger as well, as one third of the approximately 33,000 traffic accidents with fatalities per year involve poor road conditions (Pothole.info, 2023). Furthermore, the American Automobile Association estimated in 2021 that potholes alone cost drivers $26.5 billion, with 1 in 10 drivers having their vehicles damaged by potholes and needing an average repair bill of $600.

The best solution to this problem would be for local and state governments to actively repair potholes and road damage as they appear, but individual repairs are slow and governments are weary of doing repairs

due to the high cost. A survey done by the American Society of Civil Engineers found that $786 billion would be needed in order to repair the existing US roads and bridges, with $435 billion going to road repair alone. However, in 2017, federal, state, and local governments only spent $177 billion, with a focus on operation and maintenance (American Society of Civil Engineers, 2021). For response times by local governments in fixing potholes and road damage, Pima County, Arizona can be used as a general example. In Pima County, potholes are expected to be patched in 1 to 5 days for high speed and volume roads, and in 15 to 30 days for local roads. However, in actual response time, the average number of days it took for a pothole to be patched was at least 35 days, well over the estimated time for both high speed and local roads (Pima County Department of Public Works, n.d.). Therefore, it is clear the local and state governments do not see road maintenance and repair as a priority, and that there is a present danger for drivers regardless of government action.

Drivers themselves also have a potential solution; just notice the potholes and avoid them. However, this “solution” is not feasible for all drivers and is difficult to do successfully. For example, the Centers for Disease Control and Prevention found that about 12.7% of the US driving population has driven under the influence of some inhibiting substance (Centers for Disease Control and Prevention, 2022), and NVISION Eye Centers reports that about 2-3% of drivers have some vision impairment that is below the legal minimum for driving (NVISION Eye Centers, 2022). Even among drivers without impairments, reacting to potholes is difficult. According to the National Highway Traffic Safety Administration, the average reaction time while driving to a perceived threat is about 1.5 seconds in noticing and reacting to a threat. (National Highway Traffic Safety Administration, 2015). Traveling an average estimate 30 miles per hour, that lets drivers notice and react to potholes that are more than 76 feet ahead of

their vehicle. When up to 60 miles per hour, a common speed limit on US highways, this increases to a minimum 132 feet ahead. While these numbers may give the driver some space to react to potholes, the common advice in the United States is to look about an eighth to a quarter of a mile ahead while driving, or 660 feet to 1320 feet (Martin, 2022). This means that potholes can appear suddenly for drivers, as they might not be clearly visible from far away, and once they are noticed, may already be too close to a vehicle to react in time. Furthermore, as previously stated, 1 in 10 drivers on average still hit potholes despite this technique, meaning there is an issue with its effectiveness in just relying on human senses.

Given the potential dangers of road damage and potholes, the lack of work being done by governments to repair damage, and issues with human reaction times in avoiding potholes, the team decided the best solution to the problem would be to develop a mobile application to detect potholes and notify drivers faster than they would notice potholes themselves. This app would constantly watch the road, just like a driver would, but would be focused on detecting when potholes appear using artificial intelligence. When a pothole is spotted, the app would estimate its location on the road and if the vehicle is in line to hit the pothole, potentially causing damages or an accident. If the driver is in line to hit the pothole, the app would visually and audibly notify the user that they are going to hit a pothole, and which side the pothole is on. The difference between this app and the human senses based technique is that the app's AI will be trained to detect potholes at a farther distance than human drivers, and will only be trained on detecting potholes. By notifying the users well before they would spot a pothole themselves and telling them where the pothole is, the app can improve reaction times to potholes and help decrease the frequency of potholes being hit, reducing accidents and damages. The app is also planned to include extra features to notify drivers of upcoming areas with potholes. The app will

collect the pothole times and locations as they are detected and upload them to a global database. This database will then be used by the mobile app, in conjunction with the app's current location and direction, to notify users if they are approaching areas with potholes. The database will also be used with a website displaying a heatmap to show where potholes have been found recently. These extra services will assist the driver in being more alert for potholes, helping to improve driver safety, and will let local and state departments of transportation have a live updating map of where potholes have recently been found, giving them better opportunity to repair damages without having to wait on calls or spend time looking for damages themselves.

2.2 Motivations

2.2a Nicholas Gray

My motivation for this project comes from two sources: wanting to work on a project related to autonomous vehicles to help my resume and portfolio stand out and test my abilities in both creating and training a full computer vision model and implementing it into a valuable product. I've been working in autonomous vehicles and real-time computer vision projects and jobs during my whole time at UCF, but I never had the opportunity to lead an entire project myself. This motivated me to brainstorm different autonomous vehicle-related projects this June for senior design to see what I could make with my resources and connections. Initially, I wanted to work on a project related to autonomous vehicles in off-road and degraded conditions, but I soon realized that would require having an actual autonomous vehicle, which would likely be out of scope for senior design. However, while I was in Boston for the summer, I encountered New England roads, which are covered in unfilled potholes.

Thus, AI Pothole Detection was born. I wanted some way to catch the potholes before I hit them, as I was too busy paying attention to the other drivers around me and the confusing Boston road network. Thus, when I returned home, I had the idea for the pothole detection project and a real motivation to work on this idea specifically.

A side motivation for this project is that I want to write a publication about this work by the end of the project. I am considering pursuing a Ph.D. after graduation, but I want more publications and letters of recommendation before applying to a prestigious university. I already have one paper and one letter of recommendation, but if I have a project that solves a unique problem and does better than any other existing AI, that could be made into a publication that I could submit to a large conference such as CVPR, NeurIPS, or ECCV.

2.2b Kyle Williams

I’ve been interested in AI development since it went mainstream a few years ago, as I believe in its potential to process and simplify many jobs in the next coming years. I’m looking to specialize in AI when I get my masters at UCF. The idea behind AI is so sound; it takes a large swath of data and goes through a large number of iterations to find trends to guide and predict. So when I saw that there was a large number of AI projects in Senior design, I was very excited. This is a year-long project that we’ll dive deep into to get some real experience creating a multi-layered project – and the project can be the subject which I see immense potential in!

But looking through the AI projects, truthfully I found many of them to be lackluster. Many sponsors wanted to get in on the trend of using AI without going through the effort to fit them to their companies, or they had just a vague idea of what the technology even does or what its strengths

are. But I really liked Nick’s proposal for AI Pothole Detection; a live image recognition software that uses a convolutional neural network is something I’d love to work on. And so I was extremely happy to get chosen for it!

In the past I’ve had the class Robot Vision, which focuses on AI, and for my masters, I’m also in the class Computer Vision at this very moment. Getting chosen for AI Pothole Detection, however, will mark the first time that I commit to a large scale project using the technology, and the first time I will be working with many technologies we’re using in the project. Working with the car will definitely be a new experience for me, and it’ll definitely be useful for my future work. I’m excited for the challenge!

I’m excited to work with a good group of students who all seem interested in the project topic and completing the project. I have a few fears after being in some lower-quality groups in the past, but I believe in the group we have here. Our team contract is good, and I think having so many members who are in the workforce right now will give our team a great sense of momentum and an understanding of participation and deadlines. There’s nothing worse than a team member who promises something then doesn’t work on it and insists they don’t need help! We will have a difference in the amount of experience utilizing this technology and similar technologies, however. I’m glad we established early on that those who are struggling can admit it day 1 with no shame, as that will allow our members to grow and not get stuck feeling like they can’t ask for help.

2.2c Thomas Wallsmith

Creating civic infrastructure has always been an interesting concept to me ever since playing SimCity 2000 on my moms old Dell computer. Civic infrastructure has always been oddly fascinating, it's usually unnoticed but plays such a big role in our lives. Working on this project is basically working

on a marriage between civil infrastructure and ML/AI applications, two of my favorite things. I think creating a mobile app that can detect potholes would provide a lot of value to the world, especially in the realm of driver safety. If users can detect potholes before they drive into them, this system could prevent motor vehicle damage and injury to the driver. Working on a project that provides some level of public good is important to me, and makes working on this project a good fit.

The most interesting part of the project to me is the stretch goals. I think creating a database of potholes is an incredible public good, and can lead to improvements in public infrastructure by being able to view every single pothole in the city in real-time. Potholes could then be ordered by recency, allowing for specific targeting of new/old potholes for repair. I hope we make the database portion of the project a reality, but I am really excited to learn more about AI and ML systems. My background largely comes from designing and web apps, like web APIs and websites, so I hope I can learn more about those areas and add unique value to the project. I feel like I have a good grasp of resource-oriented APIs and want to use it to achieve something cool, instead of doing gig work for startups.

Overall, my motivations for this project revolve around a love for civic infrastructure and wanting to use what I've learned about API and system design to create something useful and cool for communities. I think the concept of having an AI capable of identifying potholes, combined with the concept of storing those results in a database can result in a really powerful system that could become a useful part of road quality infrastructure.

2.2d Mason Williams

I believe my motivations for this project have an arguably interesting origin story. It goes all the way back to my first semester at the University of

Central Florida. During my first semester, I decided to get involved in extracurriculars and I joined the Robotics Club at UCF. While I was there I decided to join a project titled AGV which stood for autonomous ground vehicle. That project's purpose was to build a robot that could traverse an obstacle course that had a road with lanes to follow and obstacles to avoid, including potholes. I was very excited to be on that team but I was also very naive and quickly realized I was taking out a bigger bite than I could chew. I was assigned the task of writing scripts that handled the pothole detection using computer vision. Keep in mind this was the same semester I was taking CS1 so my programming knowledge was on a very narrow scope and I had a tough course load. I tried the best I could that semester but I did not succeed in creating a working computer vision script that would detect potholes. Due to that, the task had to be handed off to someone else and I had to work on a different task for the remainder of a project. With that being said, fast forward one semester where I was enrolled for the technical elective "algorithms for machine learning". Again, I found myself in a position where I was attempting criteria that was far beyond the scope of my knowledge. I started out doing alright in the class but as weeks went by, it became more and more time consuming. It was requiring about 25 hours a week just for the coursework alone, combining that with 3 other classes and 30 hours of working at a restaurant, I could not handle it all. I ended up withdrawing from the course right when it started getting interesting. These two failures are some of the regrets I held through my academic career up until this point.

The reason I mentioned that preface was to display the fuel that drives my motivation to complete this project. This project calls for a machine learning model and some computer vision scripts to feed that model to detect potholes on the road and alert the driver. The reason I ranked this in my top 5 and requested to be swapped into this team was to give myself this

challenge again to finally redeem myself in my last two semesters. From that point in my life until now, I have gained a lot more discipline and skills to learn more efficiently and work more productively. Not only that but I have also expanded the scope of my programming skills and critical thinking. My ability to get things done has multiplied exponentially since those faults and I am ready to take on this project to prove that.

Aside from my academic regrets, I am motivated to work on this project to create a functioning and practical driving aid. I have always been interested in development for self driving and while the requirements for that scale of project is too large for senior design, this project is a good subset of that type of project. Alongside that I have always owned cars that are low to the ground and are vulnerable to damage when they hit a pothole. Even though I consider myself a safe driver, there are times when I hadn't noticed a pothole and caused serious damage to my car when I hit it. I hope to make sure no more oil pans are damaged due to unnoticed potholes. Besides those two points, potholes are dangerous and I am motivated to minimize the driving risks they cause. According to an article posted by Huesman, Jones, & Miles law firm, nearly one out of three car accident fatalities are caused by road obstructions which are mostly potholes. I am hopeful that our system will help notify drivers to minimize the risk of road obstructions, which will ultimately result in reducing the amount of driving fatalities.

2.2e Aaron Netterstrom

I was really excited to see that I was chosen to be a part of AI Pothole Detection. For a long time, my goal was to work with cars and artificial intelligence within them. Though this project doesn't totally fit the description because we will not be implementing our work into the car itself, it does give a tiny look into how all of that can work. Ever since I took an

introduction to AI class here at UCF, I have been very interested in learning more about AI and how to use it in real world situations as well as learning more about robot vision. I took an introductory class to robot vision a year ago and I was drawn to how all of that worked. I have done a good amount of research in my personal time when it comes to AI but mostly when it comes to Machine Learning with how to do something over the computer entirely, such as creating an AI and training it to play a videogame on the computer. This will be the first project that I have worked on that will include a real-world aspect.

I also enjoy the fact that this project has multiple parts to the project including the vision portion as well as the calculations that have to be done in order to see if the user will actually strike the pothole and send it to a mobile application that can alert the user if this is going to happen. I have done very little when it comes to mobile development and am also excited to see how this is all going to piece together. For work I do a lot of scripting when it comes to automating manual processes using Python, but it is always kept locally on a machine, so having all of these intertwined parts is going to really prepare me to be better at programming. I understand that with all of these moving parts and my minimal experience with a project of this size it will be quite difficult, but I am very excited that I got a project that I am interested in, and I believe that I will learn a lot and have a lot of fun.

2.2f Andrew Anchieta

Since I was in middle school, I have been very passionate about software technologies and the development of tools that enable users to improve the quality of their lives. My passion for programming began by playing video games on a PSP (PlayStation Portable). I discovered the ability to manipulate the software in the video game allowing me to obtain

significant advantages through the use of cheat codes. This eventually led to me wanting to exploit the software in the PSP to allow me to play all kinds of games through the use of emulators and software that allowed me to bypass any sort of limitations.

Over the years, I have transitioned from wanting to exploit software on my devices for personal gain over to creating or contributing to projects that allow me to contribute to the world in a positive and meaningful way. It is why I am excited and highly motivated to work with my Senior Design team on the AI Pothole Detection system for our Capstone project. A tool that serves as a driver's second set of eyes on the road which allows them to avoid potholes that can cause severe damage to their vehicles or cause a collision possibly resulting in serious or fatal injuries.

My motivation for wanting to create this AI Pothole Detection system stems from the frustration I feel when driving over massive potholes in roads that I'm not familiar with. Not to mention the amount of money spent getting tires replaced and rims fixed as a result of running over them. Monetary impacts aside, saving lives and preventing accidents is the most important and motivating aspect of this project.

With my previous programming experience with AI and machine learning concepts I feel that this project will not only provide a significant number of challenges but also bolster my skills in software development overall. My experience with Java, JavaScript, React-Native, and other development tools like Rest-API's and mobile development software will allow me to contribute to the team in developing a meaningful and possible life-saving tool for communities.

2.3 Societal Impact

2.3a Disability

This application is created to help warn drivers of potholes in the road while they are driving and although the team is not focusing on people who suffer from any sort of vision impairment or impairment in general, it will definitely help people with these problems. People who suffer from some sort of impairment while driving can struggle to notice something like a pothole or other blemishes in the road while driving, especially with the fact that the user needs to be constantly paying attention to everything happening around you.

While using the application it is this group's goal to help users avoid these potholes and give them enough warning to avoid it and miss the chance of being in an accident. Groups who suffer from vision impairment will be very much assisted in this sense who have a very hard time seeing the small discontinuities in the road in front of them.

2.3b Assisting Local Governments

One of the largest stretch goals that this group is hoping to achieve is to create a database of all of the logged potholes that could prove harmful if they were struck. Being able to have a database of potholes that is generated entirely by the users could help save a lot of time when it comes to finding these manually and reporting them to their local governments and departments of transportation in order to have the potholes repaired.

With this application the potholes will be automatically flagged and their locations will be stored seamlessly. This information will then prove to be very helpful to streamline the location and fixing of them.

2.4 Issues

2.4a Legal

The first legal issue the app has is location data. It was originally believed that the stretch goals would not have to record locations of users to record the locations of potholes. This, however, is incorrect. It is useful to link a pothole to a specific user so that the user can’t re-record seeing multiple potholes or abuse the app’s systems, which an anonymous user would have to do. The app does not have to record anything other than the location of the user. This means that it will keep the data anonymous and not link any location data to any personal information in any capacity. For the app to be operational in America and Europe, it will adhere to the privacy policies of America and Europe as guidelines for the app to follow:

In America, there isn’t currently a federal law that regulates the use, sharing, or collection of geolocation data. Legally, the app can record users' data in many states with little issue. There are a number of bills that dictate the uses of location data, but the bills are less stringent than the European location data bills. The app will follow the guidelines of the GPS act, upheld by California, Florida, Hawaii, Louisiana, Minnesota, New Hampshire and Virginia. The GPS act primarily prohibits businesses from disclosing geographical tracking data about its customers to others without the customers' permission. The app can follow the European features of the bill and have no legal issue in America.

In Europe, these are the key features that location data has to be upheld to:

- Location data should only be used when it is anonymized (i.e. it cannot be associated with any particular individual).

The app will have anonymized data.

- If it's not anonymized, the user has to give their consent for their location data to be used and accessed by this app. Also, the user must be able to withdraw their consent at any time.

The previous point is not applicable because the system will be keeping the data anonymous.

- The service provider must inform the user what location data will be collected and processed.

There will be a legal list that fulfills this requirement, and users will be able to opt out of having their location tracked. If they do not have their location tracked, the pothole data they collect will not be uploaded to a database.

- Location data should only be used for a specific reason, and only for the duration necessary for the purpose.

The app requires the use of this data to create a useful map of potholes as long as a user is driving.

The app will implement these features. It’s required by both the law and Apple’s privacy policy. The policy states: “Apps can only use location services when it is directly relevant to their services.” and also specifies “If your app uses location services, be sure to explain the purpose in your app. The app will include a privacy policy that describes how the app will be utilized such as this one:

“You consent to AI Pothole Detector to use location services for the purposes of recording pothole data for other customers.”

Another legal issue the app has to deal with is liability in distracting drivers. The app is meant to be primarily used by drivers on the road, and it

will notify these drivers if there is an upcoming pothole. To determine if the app makers can be held legally liable in this situation, one can check precedent:

- Maynard v. Snapchat: The Georgia Court of Appeals concluded that Snapchat wasn't liable for an accident that happened when a driver was traveling at high speeds and using the app's "Speed Filter." This feature captures the speed at which the phone is traveling and allows users to superimpose it on their "Snaps."

This case is similar to applicable to the app. However, the app will encourage being used while driving, instead of only being an option. The following is the requirements to bring a claim to the court over a distracted driving incident involving a smartphone:

- HQ states: "For success with a claim, the technology used must have led to the incident, caused injury and been enough of a distraction to take responsibility from the driver at least in part. This means that the duty of the tech company involved was to provide a device that was created without the express purpose of distracting a person from his or her primary focus of attention. With bright or garish graphics, these items may divert the awareness of anyone close enough to the item." (HG.Org, 2023)

The app must not distract drivers from the road with very bright or garish graphics, which is important for the design of the app. In addition, the app has a disclaimer that reminds drivers to keep their eyes on the road. Pokèmon Go implements a system when it detects that drivers are driving to remind drivers to keep their eyes on the road; the AI Pothole Detection app will as well. Without a proper disclaimer drivers must agree to, the app may have more liability in the case of an accident by the distraction of the app. With a proper disclaimer, liability would be lessened.

Additionally, the app may have liability in the case of a false positive or false negative. The AI model is not good enough to catch a pothole every single time, or to never misidentify a pothole as something else. So if a user ran over a pothole without being notified by the app, they may feel like the app has liability over not noticing the pothole on the road. The app will stress that the predictions cannot accurately identify every single pothole on the road.

Also, if the distraction of the app's notification appears when there is no pothole to be seen, the app may be liable if a user is startled, and it causes them to crash. The app will also stress that false positives can occur, and the app's notification may go off with no pothole in the road. If the app projects incorrect confidence in their model, it's more likely the app will be held liable for incorrect results. There will be multiple layers of disclaimers about the app's inaccuracies, similar to how ChatGPT has labels about the misinformation its AI network provides.

2.4b Ethical

The app is attempting to create a robust dataset of current potholes around the whole world. There are some ethical issues related to how the app is tracking the data. Users may feel a breach in privacy with the app accessing location data, but the recording of this data is completely optional. Additionally, the app is not tracking individual user's locations, only the potholes that they have seen. So, the data can't even be used as a roadmap for a user's location. The privacy implications of this are more fully covered in the privacy section.

Identifying users about potholes with the app is an opt-in service with a clear and direct way of notifying which information the app is recording and which information the app is not. Having a database of potholes is

crucial to this project, and unproblematic as long as the users understand that the app is tracking the potholes the application identifies.

2.2c Privacy

There are a handful of privacy issues to consider with this app. The app is utilizing location data, so it's critical for the app to handle the data with caution. In addition, the app uses the camera, which may lead to some privacy concerns. Finally, the app must deal with unnecessary permissions that the app may or may not need.

Location Data

Location data is essential for the app. The app will take down the location of any pothole data it finds, along with the specific user ID who found it. This requires the implementation of a database. The app is not recording any user data along with the location data, so the app will be unable to identify the specific user that the app records the data from. There are potential privacy breaches in recording exact locations of a single user across time. To avoid these, the app will take precautions against the identification of users. One precaution is a self-deleting database. The database will not stockpile pothole information that is identified. It will instead keep current and up to date with the potholes located in the world. The self deleting database will remove any potholes that have not been reconfirmed to exist within a couple weeks. For a pothole to remain identified on the map, it should have been identified recently.

When a different user identifies the same pothole another user found, the database will switch the user who confirmed the pothole's existence from the first user to the most recent user. The database will not keep unnecessary information about certain user's locations on many dates in the past.

If needed, to prevent abuse of the database, a “potholes identified” number may be added to each user’s data. This is so a user cannot abuse the measures the app has taken with privacy to abuse the system. If a malicious agent wanted to, they could identify the same pothole multiple times with multiple accounts to confirm a pothole that doesn’t exist. This will be avoided by noticing if a user has reconfirmed the same pothole multiple times in a short span of time. In this situation, the app could still detect potholes, but the user’s information would not be added to the database that has been made.

The user is not required to enable access to location services. The app’s function works entirely fine if the user does not record its data to a database. If a user opts out, then that no data would be recorded for other users to see the same pothole on the map pre-identified. The app will provide the user with data about what it’s using location services for, which is already required in the legal section. If the user sees this and would prefer not to have the app track them, then they have the option to not allow location services. All functions of the app will be available.

Camera Access

The app requires a camera. When a user starts the app for the first time, the app will issue a popup that asks for camera access from the phone. If the user allows it, they will be taken to the home screen which will prominently feature the camera on screen. If they do not allow camera access, the screen will be a black screen that tells the user they must allow camera data. The camera data that the app takes will be fed into the AI model, and if a pothole is identified, the AI data would be put on the screen as quickly as it takes the model to process the image.

Importantly, the app will not record any camera data in any capacity. The app will feed the current camera into the AI model and will not save any

camera data beyond that.. Because of this, the list of privacy issues for camera use is slim. There’s no way to access any data that’s recorded with the camera, so there’s no privacy concerns the group must take. The app will also provide a disclaimer about the camera data’s use.

Other Considerations

An important aspect of privacy with most apps is the login and user encryption. Usually, an app will utilize usernames and passwords that must be taken care of and the access of these is a privacy concern. There is no consideration with this app because there will be no login. The app will utilize the information of which device is sending the data, but there’s no way for a malicious agent to access specific user information because the app will not be asking for it.

The app will have to protect the data that it does store. This data is a user’s key and the potholes they have identified. The app must not allow malicious agents to access the database. It will have a secure server that prevents malicious attacks, and will have good measures against bots attempting to abuse the database. This is a list of some of the things that Flutter can provide the mobile app that will increase privacy:

Secure Storage of Data

- Flutter Encrypt allows the app to make the data encrypted; this is an important product for all databases

Secure Code Practices

- Testing and understanding the consequences of every line of code written and every point the user is given access to.

Third Party Library Management

- The app will be using third party software to implement the AI model. These libraries will be checked for malicious code and the app will only use libraries others have utilized.

3 Project Characterization

3.1 Goals Overview

The overall goal of AI Pothole Detection is to create an application that helps drivers avoid potholes quickly and safely, as well as assist local municipalities in collecting data about where potholes are in their local area. The user would use the app as a driving assistant, letting them focus more on the road and not having to pay as much attention to avoiding potholes. The program is meant to be easy to set up and launch, requiring little setup from the user beyond just opening the app and setting up the phone with a clear view of the road. The program should communicate pothole location clearly and quickly, and only when the pothole is an actual concern; this means that focus needs to be put on accurate detection and location estimation in order to not have the drive maneuver when they do not need to.

For the extra components of a database, advanced alert system, and map, the goal is to have these features assist in communicating further information to drivers and non-drivers about local road conditions. The app would communicate with the database to upload the potholes it detects at what time and receive any notification about upcoming potholes based on the previously logged potholes. The database would also be used to create a usable heatmap that shows where potholes have been frequently found recently, giving drivers and local governments usable information on where to avoid and where to repair roads, respectively.

We were unable to reach the stretch goal of having an advanced alert system but were able to develop a database that holds the positions of the potholes. Due to dropping the mobile application and transitioning into primarily a laptop program, the advanced alert system was not deemed to be a priority. However, the heatmap was transitioned into being a more full mapping application, with the ability to add, remove, and download reports about potholes, instead of just only being able to view them on a map.

3.2 User Stories

The following user stories were created as the main guide and focus for what development should focus on. There was a focus on giving priority to the end-user (drivers), while also giving some focus so as to make later development easier.

- As a driver, I want to get on the road quickly so I can get to where I need to go.
- As a driver, I don’t want to be spooked by sudden load noises, as that will make me possibly jerk the wheel.
- As a person, I want my personal information safe and not randomly logged, as what would be the need for a pothole detection app to know who I am?
- As a developer, I want to be able to iterate quickly and add new features, as that will give drivers more information.
- As a government worker, I want to be able to drive around a large area to collect potholes, them view them later on a large map.
- As a government worker, I want to be able to collect data on the potholes collected and know where they are so that they can be patched at a later time.
- As a citizen, I want to be able to view where potholes are in my local community, so that I may avoid them and better understand how well

my government is dealing with the issue.

4 Requirements

4.1 General Requirements

- Minimal setup difficulty
    - Users should not have to spend more than 5 minutes setting up the app for use. While there will be some needed setup time in order to set up a phone stand for the phone to watch the road, starting the app should be as simple as opening the app itself.
- Use minimal hardware resources
    - The system should have to use as few pieces of additional hardware as possible and use what hardware it is using as sparingly as possible. Here, this means that the app needs to not be heavily demanding on the phone hardware, especially for older devices.
- Fast and Communicative
    - Notifying users needs to happen as quickly as possible and only when there is importance in communicating an approaching pothole. Potholes that won’t be hit should be ignored and potholes need to be verified in some fashion with consecutive detections.
    - Alerts also need to be non-startling, so as to not frighten the driver and cause them to drive unsafely.
- Protect User Privacy
    - The app and related systems should not log any personal or identifying information. If any information needing an ID is logged, the log should be anonymized and not traceable back to

a specific device or person.

- Well documented
    - All code and documentation should be easily readable and well-documented. AI related work needs to be tracked and training results recorded, and mobile app development should include clear code comments, modular software design, and external diagrams and documentation of app structure.
- Feature Expandable
    - The app and related features should be constructed to allow for later iteration and expansion, especially for the later stretch goals that will add further features.

4.2 AI Requirements

- Detects Pothole
    - The model must detect potholes specifically. While other road damage could be included later, the specific focus on the model is in detecting potholes.
- Accurate Detection
    - The pothole detection model needs to be able to detect potholes accurately and consistently. If it cannot do either of these, then the model is performing poorly.
- Fast on Edge Hardware
    - The model needs to be able to run on portable hardware in real time. Given the diversity of smartphone hardware, this can be a serious limitation. Focus needs to be put on selecting a model with a design on real-time performance and high accuracy regardless.
- Work in Any Environment

    - A pothole detection model cannot work in only one area. If it only worked in certain environments (for example, higher end US neighborhoods where there are rarely any potholes), then the model would have a clear bias that would not assist all users.
    - The dataset the model is trained should be diverse in location and road quality. This will both help reduce bias and make the model more robust.
- Verified Detection
    - The model must include some kind of verification system to determine that the detection is reliable. This will likely be a threshold on each detected pothole to only include confident detections.
    - In order to ensure potholes are properly identified as they come closer to the vehicle, a tracking algorithm will be needed to check that detected potholes are real. This will also be used to verify their estimated location and if they are in line to be hit by the vehicle.
- Small Storage Size
    - The model should have a small storage size in order to minimize the installation size on the smartphone. This will include choosing smaller models and performing optimizations to shrink model size further.
- Accurate Benchmarking
    - In order to compare the performance of different AI models, a standard set of measurable benchmarks to select the best performing model.
    - The benchmarks must be relevant to the type of AI model being used, depending on if classification, detection, or segmentation models are used.

- Training Data
    - The training data, on top of being diverse, must be sizable enough to allow for fine-tuning and, if necessary, raw training of any AI model. Larger datasets will allow for more robust training.
- Portable to Mobile
    - Any model must be able to be ported to the mobile app regardless of the framework used. This means the model should not be limited to running only on Android or IOS.

4.3 Mobile Requirements

- At Least Three Pages
    - The app should at least include a settings page for user adjustments, an information page to give users a tutorial and further information about the app, and a main page that will show the phone’s view and display alerts.
- Include Tutorial
    - The app should include a tutorial page or interactive tutorial to teach the user how the app works and how it should be set up.
- Camera Permissions
    - In order to run any AI model, the mobile app will need permissions to access the phone camera. This access will be used while the app is open to watch the road in real time for the AI model.
- Understandable Alerts
    - Visual alerts must be displayed on the mobile application with clarity such that the driver should be able to understand where the pothole is without looking directly at the screen.

    - Auditory alerts must clearly state the pothole location in an understandable and calm voice to minimize time needed to understand the message and to not scare drivers.
- Adjustable Settings
    - The app should have a settings page that lets the driver adjust the alert volume and visual indicator opacity to user preference and accessibility needs.
- Integrate with AI Model
    - The mobile app should be able to integrate with an AI model directly, without the need for a cloud API. This is required due to the potential latency costs for using a cloud API in real-time,and the potential for low quality or no data reception.
- Fast and Stable Performance
    - The app overall must not be slow, laggy, or possess any bugs that can cause a crash during operation.
    - The mobile app and AI model together must not cause the app to run slower than real-time. At a frame rate of 5 FPS, the app and model should not take longer than 200 milliseconds to process an image from the camera for detecting if it has a pothole.

4.4 Laptop Program Requirements

- At Least Three Pages
    - The app should at least include a settings page for user adjustments, a mapping page for displaying where the potholes are on a map, and a main page that will show the phone’s view and display alerts.
- Camera Access

    - In order to run any AI model, the program will need t o be able to access a camera or video stream of some kind. This access will be used while the app is open to watch the road in real time for the AI model.
- Understandable Alerts
    - Visual alerts must be displayed on the mobile application with clarity such that the driver should be able to understand where the pothole is without looking directly at the screen.
    - Auditory alerts must clearly state the pothole location in an understandable and calm voice to minimize time needed to understand the message and to not scare drivers.
- Adjustable Settings
    - The app should have a settings page that lets the driver adjust the alert volume and visual indicator opacity to user preference and accessibility needs.
- Integrate with AI Model
    - The mobile app should be able to integrate with an AI model directly, without the need for a cloud API. This is required due to the potential latency costs for using a cloud API in real-time,and the potential for low quality or no data reception.
- Fast and Stable Performance
    - The app overall must not be slow, laggy, or possess any bugs that can cause a crash during operation.
    - The laptop program app and AI model together must not cause the app to run slower than real-time. At a frame rate of 5 FPS, the app and model should not take longer than 200 milliseconds to process an image from the camera for detecting if it has a pothole.

4.5 Miscellaneous Requirements

4.45a General Requirements

- Additive Improvements
    - The stretch goals should be additions to the app, not core requirements. If they are required in order to make the app run at its core (live pothole detection), then they are not stretch goals.
- Provide Useful Addition
    - All stretch goals should provide a change that improves either the experience for drivers or adds experiences that widen the potential user base to non-drivers. However, stretch goals should be focused with drivers in mind.
- Equal Development
    - Stretch goal developments should be given the same effort as the core application is given. If a stretch goal is chosen to be developed, then it is confirmed to be part of the application.

4.5b Database Requirements

- Store Relevant Pothole Information
    - The database needs to be able to store pothole locations and the time the potholes were found.
- Accessible by Outside Applications
    - The database needs to be accessible by the mobile app and heatmap website to allow for uploading and downloading relevant data for their use cases.
- Uptime
    - The database needs to have constant uptime with no crashes or significant downtime.

    - If there is downtime a system must be set up in order to allow for any missed uploaded potholes to be added to the database.
- Fast Updates and Syncing
    - The database should update quickly when new uploads are added to the database and any exports of the table onto mobile devices should sync with the database to provide users with accurate tables.
- Database Size Management System
    - In order to minimize the storage size of the database, potholes in nearby locations should be clustered together as an average location and a count of the number of potholes in that location.

4.5d Map Requirements

- Web Accessible
    - The heatmap should be accessible using a website or web application to make access universal and not require any installation.
- Desktop and Mobile Friendly
    - The heatmap should be accessible on both desktop and mobile devices to allow any user to access the heatmap, no matter what device they use.
- Accurate with Database
    - The heatmap should show the data as the database is currently reporting. The heatmap system should build directly from the live database and use the most up to date data.
- Shows Local Area
    - The heatmap should center around the user’s local area so that the map does not start at a location that is not relevant to the

user. This can be done without violating identity privacy concerns by using IP addresses.

- Accessibility
    - The heatmap should have accessibility features that allow any user to use the heatmap system without impairment.
- Relate Locations to Road Maps
    - The heatmap should report directly where potholes are and when they were last reported while showing what roads they are on. This gives users the most useful information without providing unneeded information.

5 Concept of Operations

Based on the general requirements, the project was originally envisioned as a mobile application that drivers use to notify them of incoming potholes automatically before they notice themselves, giving them more time to move out of the way of the pothole. There are two major components to the application: the pothole detection AI and the user notification system. The pothole detection AI will use the smartphone's existing processor and camera to watch the road and detect any incoming potholes in real time. This fulfills the first two general requirements of using very few hardware resources, as the application only uses a driver's smartphone, and requires minimal setup, as the user only needs to place the smartphone in a dashcam view. However, some may note that making the driver set up a smartphone in a dashcam view is not a minimal setup, as it forces drivers to have extra hardware to set up the phone to the correct position in the first place. However, it has been decided that, given the need for there to be some camera setup and the requirement of minimizing extra hardware, making the driver have a phone stand is not a great inconvenience compared to other alternatives, such as having a dedicated

device with additional hardware for this task. Figure 5.1 shows the overall concept of operations for the phone's needed field of view and operation viewpoint from the user.

Figure 5.1: Basic Overview of Operation.
Left: Phone's Estimated Required Field of View
Right: Conceptual Smartphone Location and Alert System
Sources: Vecteezy.com, Clipground.com

For notifying the user, it is envisioned that the system will have some visual and auditory alert to notify drivers immediately when a pothole is detected, but only when the driver is on a path to hit the pothole. This extra threshold for notification exists because notifying the user of every pothole may cause unsafe driving behavior or lead to the driver ignoring the warnings, defeating the purpose of the application. By using a simple visual or auditory cue, the driver should be able to understand which side the pothole is on more quickly, and knowing that they are on a path to hit it, make the necessary actions to avoid it. There is a concern, however, that by making the driver trust the system and not see the potholes themselves, they will oversteer, which may cause greater harm than just hitting the pothole. For this scenario, it is also envisioned to have a highlight around the detected pothole so that the driver can understand where it is and thus make a safe maneuver to avoid it.

For the current version of the program, there is now a laptop application instead of a mobile application. However, the vision of the

program is very similar to the mobile application. The driver will set up either their smartphone or a camera of some kind inside the vehicle, which is connected to a laptop or other higher power machine. The driver then beings driving, and there is a laptop program that operates in a similar fashion to the original mobile application.

For the stretch goals, it is envisioned that the mobile app will log the detected potholes onto a cloud database, using the phone's current GPS coordinates and timestamp to log the potholes detected. This database will then process and group the potholes together to minimize storage size and to simplify the two main stretch goals: an advanced alert system and a viewable heatmap. The advanced alert system is envisioned to operate in a similar manner to the live alerts, where there will be a visual and auditory alert of the driver approaching an area with potholes. However, unlike the live alerts, this alert will appear on the top, and will only indicate that potholes are approaching. In order to retrieve these potholes, the database would be wrapped with a back-end API that is called on some set interval with a phone's current direction and location. The API would take the direction and location, query the database for any nearby potholes, and return either a location if the driver is approaching potholes or no value if not. These potholes in the database would have an expiration date if they are not detected in some time. This would be done in order to remove potholes that have been patched by local governments, although this may also occur on local roads rarely traveled.

For the map, the concept of operation includes a website similar to Google Maps or OpenStreetView with map markers of where potholes have been recorded in the database. This map would be accessible on both desktop and mobile browsers and would initially center based on the user's IP address, as it is expected most users would want to check their local area. The map would allow for moving and zooming on the map and would show

the specific locations of potholes when clicked on or hovered over with a mouse. It is also envisioned that there would be extra controls for the map related to filtering potholes based on dates to see where potholes have been recently found. This feature would be of extra use to local governments, as they could use this filter to find where potholes have been recently found and need to be repaired. Finally, there is envisioned some ability to add and remove potholes from the map for government workers, allowing them to maintain the database directly through the map, instead of on the database itself.

6 Division of Labor

6.1 Timeline of Roles

Stage 1 (September 2023 - December 2023): Research and Making AI

- AI Modeling and Data: Nicholas Gray, Mason Williams
- AI Modeling: Kyle Williams
- AI Data: Thomas Wallsmith
- Mobile Dev: Aaron Netterstrom, Andrew Anchieta

Stage 2 (September 2023 - January 2024): Mobile App Development and Integration:

- AI Modeling and Data: Nicholas Gray, Kyle Williams, Thomas Wallsmith
- Mobile Dev: Aaron Netterstrom, Andrew Anchieta, Mason Williams

Stage 3 (February 2024): Database Creation and Management

- AI Development: Nicholas Gray, Kyle Williams
- Database Setup: Thomas Wallsmith
- Rest API: Aaron Netterstrom, Andrew Anchieta
- Version Control and Documentation: Mason Williams

Stage 4 (February 2024 - March 2024): Mapping Integration other Stretch Goals

- Website: Mason Williams
- Mobile Development: Andrew Anchieta
- AI Development: Nicholas Gray
- Rest API: Everyone Else

Stage 5 (FMarch 2024 - April 2024): Emergency Fixes

- AI and Laptop Program Development: Nicholas Gray
- Website: Mason Williams, Thomas Wallsmith
- Rest API: Everyone Else

6.2 Individual Contribution

6.2a Nicholas Gray

My main responsibilities were related to being project manager and being the head of the AI development team. I worked on exploring the problem space and creating the problem definitions and requirements that would guide the development of the project. I researched what previous works have done in terms of dataset, model selection, and setup for actual use. I also researched the pros and cons for the different types of models there are for pothole detection, and the different options in libraries for developing the models.

Besides research, I explored the existing pothole datasets and created our own pothole dataset by merging some previously found datasets. I researched and set up the data storage systems for training our AI models and oversaw distributing the workload of merging the datasets amongst the AI team. I also provided information to the team on how computer vision and artificial intelligence works in general and in terms of pothole detection.

For the mobile team, I assisted in finding relevant libraries to integrate the produced AI models into the mobile application. I researched and listed libraries for Flutter and React Native that should be able to work with any model exported from PyTorch or TensorFlow. I also gave my thoughts on the app design, focusing on how to maximize understanding in the initial user tutorial so that they would not have to reference the tutorial again later. My idea was to have an interactive tutorial that would show the user directly how the app worked and what each page in the app did, with the information page allowing for the tutorial to be run again if need be.

I did all of the development for the laptop program. I trained both the MobileNetV2 and the YOLOv8 models we used in the mobile application and the laptop program, respectively. For the laptop program, I wrote it entirely myself, and set up integration for both the smartphone camera stream and webcam. I researched and developed the lane detection and setting systems, and developed the laptop mapping program myself. I also researched the available reverse geocoding libraries for both the laptop program and the website front-end.

Finally, I was the SCRUM master for the team and the one who tracked deadlines. I managed the Jira board, the team timeline, and made sure the team had scheduled for the demo and final committee.

6.2b Kyle Williams

The main part in this project I worked on is related to all aspects of the AI model and its execution. I was one of the two members in the AI development team, and I retrieved databases we trained the model on, I combined the datasets, and filtered the datasets for images with potholes in them. I trained AI models on the dataset, including YOLOv8, YOLOv5, and Roboflow 3.0 models, as well as writing the code for a Pytorch model. I also

researched and decided which models are worth training for the sake of comparison, researched which library most efficiently runs the AI models for integration into Flutter, and assisted a little with the mobile pipeline. I trained my models using Jupyter Notebook, and applied preprocessing to the dataset before training models with differing number of epochs to find the best model.

I also applied varying augmentations, such as crop, blur, noise, rotations, exposure, and hue alterations. I uploaded my models online, to Roboflow and the weights to Drive, for ease of access for my team. Comparing them, I found the best model was using YOLOv8, so we used YOLOv8. I tested the models against one another in live testing, both testing the models on videos of potholes in dashcam position using Ultralytics, and bringing my laptop in the car to potholes around UCF using a hotspot to have it identify them.

I attended every single meeting except one I missed for a cruise, created the CDR slides, and I was a core contributor to many of the idea implementations for the app and laptop program, such as the pothole's expiry from the database, grouping of close potholes together, and a delay on the identification from the app. I really cared about this project and making it useful and successful, and I think I crushed my half of the project.

6.2c Thomas Wallsmith

Across the project's lifetime, I mainly had two roles. At first, I helped with data collection and structuring the project. I looked through some data sources and filtered based on the given criteria.

Soon after, I moved to API and Database design. I designed the underlying database tables and resources based on the needs of the group. This required reaching out to all teams and figuring out something that

worked for everyone. After designing the Database, I built the API to interact with it, allowing for coordinates to be stored using PostGIS. I also managed deployments and aided in building the web app. I set up the deployment with Render and aided with the web app deployment as well.

6.2d Mason Williams

My role throughout this project changed multiple times due to restrictions and the needed workload of certain teams. The first role I took on which didn't really change was the role of being project co-lead to help Nick with administration and infrastructure. That role entailed keeping the mobile team organized and setting up meetings and ensuring everyone was aware of important dates. As well as keeping the jira workflow moving.

Aside from that role, I was responsible for designing the mobile application architecture. I was supposed to be responsible for the model integration on mobile but when I created my dev environment and attempted to test the foundational app, I was unable to simulate the app and properly test my changes. Therefore I was only responsible for just the design but not any implementation to do work tool restrictions.

Due to me not being able to work on mobile anymore, I then switched to the client side portion of the web application. I was responsible for styling the front-end and ensuring the google maps plugin was working properly as well as all the export features required.

6.2e Aaron Netherstrom

For this project I will be entirely focusing on mobile application development. I will be working alongside Andy and Mason to design and create the mobile application as well as implementing the machine learning model into it. I have never worked on an application before and that is part

of the reason I had picked this role. As I mentioned above, I have worked with machine learning as well as computer vision before but other students in my group really wanted to explore this part of the project and I wanted to have everyone learn a lot through this project. If I chose to be on the team that focused on the model, I feel like I would not have been able to broaden my horizon when it comes to software development. I feel like it is much more important to be exposed to all aspects of software development including the creation of the model, the mobile development, and the web-based development.

With mobile development being the only aspect of the project that I have next to zero experience in, I thought I would get the most out of this class by being a part of this team and being a more well-rounded programmer. For the first weeks of this project I conducted research into multiple different types of application development frameworks our team could use and ultimately, we could choose the best one that will work for our project. After this I was able to create a mockup skeleton of the application in Figma and then set up the framework on my personal system.

I ran into multiple problems with how my machine is set up and I had to create a new local user in order to get it all functioning properly. Now I am currently working on setting up a skeleton of the application. Once the skeleton is complete, this will allow the entire mobile team to work on separate pages of the application at the same time with minimal conflicts when pushing to main.

After helping with mobile development I moved towards helping setup and manage the database. This consisted of helping plan and lay out the foundation for the database with what we are going to store and helping problem solve on how we would solve certain problems when it came to

using endpoints in terms of creating/incrementing new potholes and the bounding box endpoint to return potholes in a given area.

I also created the initial commit of the front-end web application that displayed a map using the Google Maps API. I was able to obtain a API key for the google platform as well as setup the custom map and create a base for the web application to flourish on. I was able to implement the map on the website as well as add very basic styling to help view the web application better.

Finally, after the application had been completed I researched the easiest ways to deploy our project because we were not using any sort of framework for the web application. I came to the conclusion of using Amazon's AWS Amplify which allows for very simple "drag and drop" deployment of our web application.

6.2f Andrew Anchieta

In this project, my main focus will be on mobile application development. Since I have had experience with mobile development with React-Native, my team and I decided I would be one of the best candidates for the development of the mobile application. As a result, the project group is split up into two main groups – AI and Mobile. Being on the mobile team, I have made contributions to the project related to the mobile application.

The very first thing I had done was fixing the design of the mobile application after the basic backbone structure was created by another team member on Figma. Since the design of the app was mainly focused on the features, I focused on making the design "pretty". I added more colors, icons, and blur layers in the user interface while making sure the warnings were brightly colored to catch the driver's attention and the text font and size remained consistent throughout the app.

The next contribution I had done was getting Flutter SDK Framework and libraries installed on my local machine. While it was not an easy process, eventually after getting all the dependencies like Android Studio and CocoaPods installed, Flutter worked like a well-oiled machine. Running the command “flutter create ai_pothole_mobile '' into the terminal on my preferred directory location created the basic app needed to get the mobile application started. I tested the basic app on an iOS simulator using Xcode and on an Android emulator using Android Studio.

The next step after creating the app was having it pushed onto our GitHub repo. After setting up each individual group members’ Flutter environment for their local machine, between the mobile development group I, we cloned the app on our local machines, and using the Flutter app I created, I copied the contents as a subfolder of the main cloned GitHub repo, committed changes, and pushed it to our repo. We set new branches for each of the members of the mobile development team for app development. This marks the beginning of mobile app development.

My goal for the app development is to have a usable product as close, if not better, to the original design as possible. While developing the app pages is assigned for another team member, I am trying to find the most efficient and easiest ways to implement features to the app, like a navigation bar and camera view. Other ways I can contribute to the app development is making sure the design of the user interface is as close to the original and as easy to use and read as possible.

While my main focus is on mobile application development, I will find other ways to contribute to the project. Since I also have experience in backend/APIs and testing them, I will deal with that area if it ever comes up.

7 AI Research

As per the AI model requirements, the AI model needs to run quickly and use as few hardware resources as possible. As the model will be in a vehicle environment, the AI model selected needs to be real-time, which means that the model needs to be able to perform and complete a task within a specific time interval (FutureLearn, n.d.), which in computer vision AI is usually less than a second. Due to these requirements, consideration had to be put into the programming language, deep learning library, model architecture to use, and the sensors that will be used to support the mobile app in detecting potholes.

7.1 Programming Language

There are multiple different programming languages that are used in AI and computer vision research and development, each with their own focuses and specialties in development, with different levels of support for artificial intelligence development. Below are four popular languages for AI development, going from most frequently to least frequently used in industry.

7.1a Python

Python, a high-level general purpose programming language was created on February 20th, 1991 with a focus on being Pythonic, or in general terms, readable and easy to program (Kuhlman, 2011). Python is one of the most popular programming languages, with over 48.07% of respondents in Stack Overflow’s 2022 survey using the language. (StackOverflow, 2022)

In terms of AI and computer vision development, Python has one of the largest libraries for data science, model development, and computer

vision out there. For data science and processing, the most popular libraries are NumPy, SciPy, Matplotlib, and Pandas. NumPy is an open-source library that enables “numerical computing” on Python, which allows for array and statistical functions to be performed much faster in Python by using Python as a wrapper around C++ functions (NumPy, n.d.). SciPy is an algorithm similar to NumPY with a focus on scientific and technical computing, given implementations for multiple algorithms and techniques with the same C++ core implementation as NumPy. Matplotlib is a plotting library that lets users generate unique plots for their data with a large amount of control over how things are visualized and graphed. Finally, Pandas is a library for ``manipulating and analyzing tabular and time series data, and gives users a large number of options for performing data analysis.

For AI development, Python has two main options: PyTorch and TensorFlow. Both of these libraries are popular in AI development and research and are used to design, train, and package AI models to be used in a number of applications. Both of these libraries can work with NVIDIA CUDA, allowing training to be accelerated by performing operations on the GPU, which excels at the matrix operations modern day deep learning AI requires. For computer vision related tasks, the popular computer vision library OpenCV has both a native C++ implementation and Python bindings, allowing the library to be easily used in conjunction with other libraries.

However, Python is a language known for having issues with parallelization and speed. When using native Python code and comparing to C, Python takes between 20 to 100 times longer to run common algorithms, on average (The Computer Language 23.03 Benchmarks Game, 2023). However, given that many Python libraries are wrappers around C++, and there exist libraries such as Numba that allow for Just-in-time (JIT) compiling, this speed issue can be minimized and allow for efficient processing and calculation.

7.1b C++

C++ is another high-level general purpose programming language created in 1985 by Bjarne Stroustrup as an extension to the C programming language to include object oriented programming capabilities (Stroustrup, 1997). Unlike Python, C++'s language is very similar to C, and demands more setup from the user, including memory management, specific variable classes, and all code operating inside of classes and functions. Furthermore, while Python is primarily partially compiled and then interpreted, C++ is completely compiled, allowing for much faster calculations. C++ is commonly used in edge and embedded software development, which given the need for a fast and minimal power cost AI model, may be appealing.

For AI related support, TensorFlow does have a Lie version for microcontrollers, and other packages such as Caffe and mlpack allow for deep learning development to be done in C++ itself (Chatterjee, 2020). OpenCV, as stated previously, is also natively supported on C++, meaning that it should be able to support the same type of development as Python. However, given the extra syntax complexity compared to Python, iterating and altering models and training schemes is more time consuming, which may not be desirable for fast development.

7.1c Julia

Julia is a relatively new programming language released in 2012 to be a high-level, general purpose dynamic programming language with a focus on numerical analysis and computational science (Bryant, 2012). Julia is distinct from other languages in that it is designed to be used as a scripting language with a shell prompt for inputting commands. Many see Julia as a competitor to Python and C++ in AI and computational mathematics due to

its efficient garbage collection system and just-in-time compiler (Karpinski, 2013).

For AI support, Julia has its own list of supported packages, with Flux.jl, MLJ.jl, and Knet.jl being the most popular (JuliaHub, Inc., n.d.). These packages have a unique format compared to PyTorch and Tensorflow, as they involve more explicitly connecting each layer of the network, unlike TensorFlow to some extent, while still abstracting some of the training functions from the user, unlike PyTorch. For OpenCV support, it does support Julia bindings, however there is a requirement that Python is already installed on the machine.

Julia's main downside is its lack of widespread adoption and support. In the 2022 Stack Overflow survey, only 1.53% of users used Julia, which is a sharp difference between Python's 48.07% and C++'s 22.55%. Furthermore, its main differences and offering lean more towards supporting researchers and scientists and not directly AI development and deployment. Given that the project is mainly concerned with model training and deployment onto an edge device, Julia may not have the offerings that are required for this project.

7.1d Mojo

Mojo is a recent programming language released in 2023 that is meant to be a superset of Python, providing a more unified framework for artificial intelligence development. Unlike Python, which does not natively run efficiently due to its partial interpretation of code versus compilation, Mojo compiles the Python code and includes a runtime environment internally for faster code execution (Modulor Inc., n.d.). Mojo is also designed to perform parallel processing of code, unlike Python, which performs only single-threaded execution. According to Mojo's own claims, it can run 68,000

times faster than native Python, and 5,000 times faster than C++. Mojo is also able to interface with existing Python libraries, such as Matplotlib and NumPy, making it an appealing option for optimizing model performance and development.

However, the main concern with Mojo is its maturity. Since it was released in 2023, it is not fully publicly available yet, and is still in a preview release. Furthermore, by being such a new language, there are not many existing support groups or existing guides to programming in and debugging Mojo code. This is unlike the other languages, which have had time and experienced developers to develop support tools and documents for their languages. Finally, given that the existing Python libraries already have techniques for improving performance, Mojo's claimed improvements may not be greatly needed for this project.

Given the options listed above, Python was chosen to be the programming language for AI model development. While other languages may offer faster processing natively, the extra development time either with the syntax or lack of existing community represent a major hurdle for development. Furthermore, speed issues in Python have been addressed with the existence of the previously mentioned libraries, most of which are Python bindings around C++ libraries. Finally, Python's large community and large unofficial and official documentation in AI and computer vision make it stand out amongst the other languages, making it the best choice for efficient development.

7.2 Deep Learning Library

In deep learning development on Python, there are two main libraries used for development: TensorFlow, owned and developed by Google, and PyTorch, initially created by Meta and now owned by the Linux Foundation.

Both libraries have different advantages and disadvantages for development, and thus, it was important to choose one library for the project and stay with it. The choice of library would affect the availability of using existing repositories and how the model would be deployed to the mobile application.

7.2a TensorFlow

TensorFlow is a popular deep-learning library for making production AI and allows for quick design, training, and deployment of AI models. Its design uses an API-style construction for its neural network models, where models are constructed by creating a single object with the necessary layers and letting TensorFlow control the forward propagation, optimization, and scoring during training. TensorFlow's main advantage is being able to quickly make simple models and deploy them, especially on mobile with Tensorflow Lite (Google, n.d.).

However, Tensorflow is disadvantageous for making more complex models that involve different layers skipping each other and rejoining, as its API-style architecture lends itself more to models with no skip connections. Furthermore, TensorFlow's design does not make it easy to make quick changes deep in the architecture, unlike PyTorch, as TensorFlow will change entire layers and pipelines. Finally, TensorFlow is not structured in a Pythonic style, meaning developing in TensorFlow is similar to using an entirely new language.

7.2b PyTorch

PyTorch is an object-oriented style deep learning library that is more popular in research environments and developing state-of-the-art models. PyTorch's syntax is very similar to the regular Python syntax, so it is easier to understand and learn for those already familiar with Python compared to

TensorFlow. PyTorch also allows for faster testing and modification, as its dynamic computation graph allows for immediate feedback on any errors found, while TensorFlow uses a static computation graph that makes it harder to debug. PyTorch is also the library of choice in computer vision research and development, as computer vision is a heavily researched field, so there is some blend between industry and research tools and developments in this field.

The main disadvantage of PyTorch is that it has a higher learning curve and has greater demands on the developer writing it. Unlike TensorFlow, PyTorch requires the developer to design the feed-forward component of the network alongside the network layer definitions. Furthermore, PyTorch does not have automated training, unlike TensorFlow, so the developer has to write the code to feed the training data to the model, calculate the loss, and backpropagate the gradients to improve the model. Another issue is that PyTorch's ecosystem and edge deployment systems are not as fleshed out as TensorFlow's. TensorFlow is widely considered to have a larger and more robust ecosystem compared to PyTorch, and while PyTorch does have some capabilities to export its models to edge and mobile applications (The Linux Foundation, 2023), it is a more involved process and not as smooth as TensorFlow.

Based on the advantages and disadvantages between these two libraries, it was decided that we would use both TensorFlow and PyTorch is our trainings where appropriate. PyTorch is better used in the laptop application, as you can directly interface with the GPU and it is easier to set up training and inference. For mobile training, on the other hand, it is better to use TensorFlow, as there are better mobile deployment options and there

is the TensorFlow Object Detection API for directly training mobile oriented object detection models.

7.3 Task Selection

In order to detect potholes, it was important to select which type of task the model would perform. Detecting potholes is a broad task that could be perceived as just classifying if there is currently a pothole in view or finding where a pothole is currently in view in an image. Furthermore, if finding a pothole in an image, it is important to decide what level of detail is needed, i.e., if it is important just to know the general area of the pothole in the image or if knowing the exact pixels in the image corresponding to the pothole are important. Based on these possibilities, there are three possible ways to define the tasks: image classification, object detection, and object segmentation.

7.2a Image Classification

Image classification is the most straightforward task of the three; the model would simply have to decide whether or not there is a pothole somewhere in the image without giving any information as to where. Image classification from an architecture viewpoint generally involves feeding an image into a convolution neural network that eventually feeds into one or more output nodes, where each node is a potential class. In this case, it would be a single node for whether or not there is a pothole in the image. Figure 7.2a.1 shows an overview of image classification, where a whole image is binarily classified as having a pothole or not.

Figure 7.2a.1: Image Classification Overview. Source: Clipground.com

For detecting if there is a pothole on each side of the road or in the lane of a tire, the raw image could be cropped, and the cropped image fed in for classification. However, image classification does not provide information on the pothole's location and does not let the AI confirm if one pothole is the same as a detected pothole in a previous image. Furthermore, classification does not provide any information on the number of potholes in the image, which would interfere with the need to submit the number of potholes found to the database.

7.2b Object Detection

Object detection is a more complex task than image classification, which involves not only classifying if there is an object in an image but where that object is in the image as well, usually in the form of a bounding rectangle around the object. Object detection is a very popular computer vision task, as locating objects in images is useful in fields such as security, transportation, healthcare, and agriculture, to name a few. Furthermore, object tracking, which involves assigning IDs to bounding boxes detected in images coming in sequentially, allows for objects to be tracked as they go through videos, allowing computer vision to be used in real-time tasks where continuous information needs to be understood.

The basic operation of object detection is similar to image classification, but instead of just returning a single classification for the entire image, it splits the image into small sections and gives a classification and confidence to each section. Then, the model generates a list of potential bounding boxes that could have something inside. If there is a bounding box where the sections contain majoritively one class, then that bounding box is kept and given the majority label. Finally, all labeled bounding boxes are thresholded by a minimum confidence, and the model outputs the bounding boxes with the highest confidences. Figure 7.2b.1 below demonstrates how object detection is performed, with a bounding box drawn around the detected pothole.

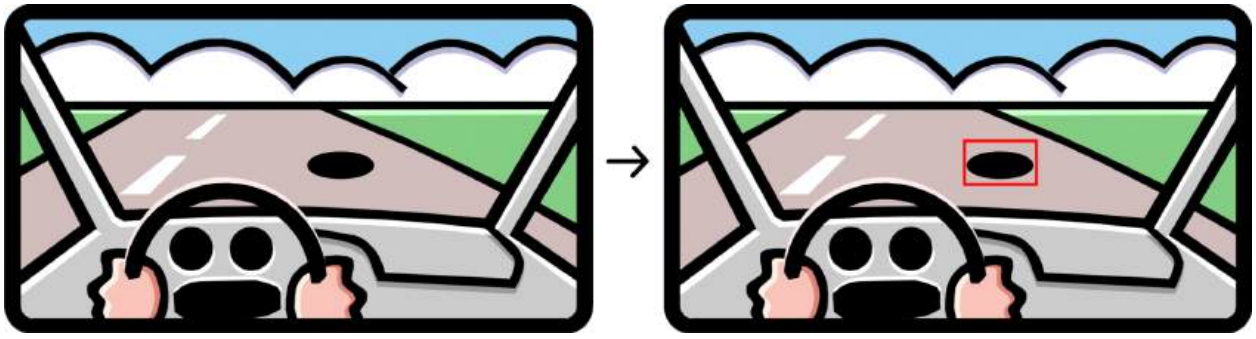


Figure 7.2b.1: Object Detection Overview. Source: Clipground.com

For this project, object detection is very useful because locating and tracking detected potholes will allow the application to estimate where the pothole is in the real world and if it is in line with the driver's vehicle's wheels. Furthermore, having the potholes' locations allows for a more accurate estimation of their locations, as there are models that can estimate the depth of a location in an image, which can be used to calculate the potholes' distance from the phone. Finally, there are many real-time object detection models widely available for edge and mobile computing, so object detection could be a good task definition for the project.

7.2c Object Segmentation

Object segmentation is similar to object detection in that the model finds the actual location of an object in an image. However, unlike object detection, object segmentation finds the object's outline and returns it as a polygon. The particular type of object segmentation necessary for this task is instance segmentation, which involves only finding the outline of important objects and ignoring the rest of the image. Object segmentation models operate like object detection models, but after labeling the bounding boxes, there is a mask generation layer where each of the pixels inside the bounding box is classified as part of the object inside or not. After the pixels are classified, an outline mask around the positively labeled pixels is generated and returned by the model instead of the bounding boxes. Figure 7.2c.1 below shows instance segmentation for pothole detection, with an outline around the detected pothole shown.

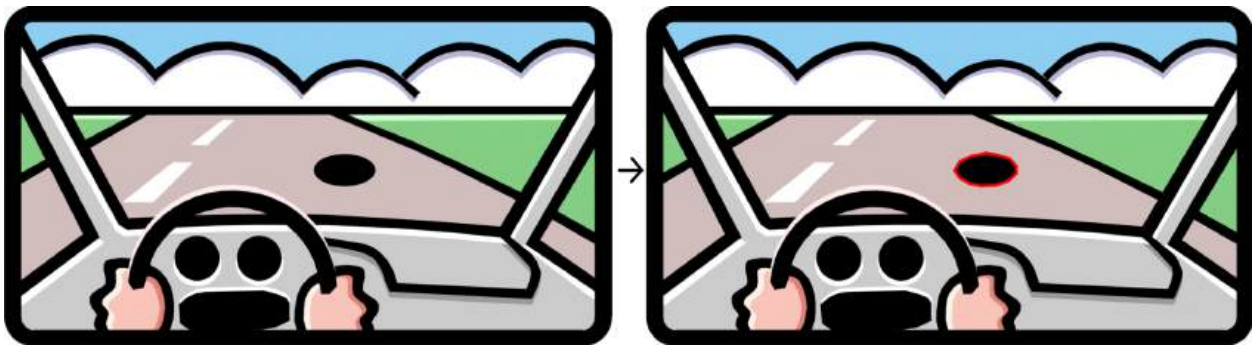


Figure 7.2c.1: Object Segmentation Overview. Source: Clipground.com

A major advantage to using object segmentation models is that it gives a more precise definition of where the pothole is located in the image, which can allow for more accurate location estimation compared to bounding boxes. The nature of potholes is that they are not always perfectly circular, so a simple bounding box could cover much more area than the actual pothole covers. Furthermore, highlighting the pothole itself may

communicate more information to the user than just a simple bounding box, as the user would still have to look in the bounding box to find the pothole.

However, there are downsides to object segmentation that make it not desirable for this project. While object detection does provide more information, it comes at the cost of having a more complex model. Object segmentation models are known to perform slower than object detection models, given the need for an extra stage to refine the outline masks of objects. Given that this model will be used in real-time and potentially high-speed areas, this performance increase is of great concern. Secondly, the extra information may be of little value to this project. While having more accurate calculations is generally desirable, in this project, having a good estimation while keeping up performance is a minor loss overall. Finally, while drawing object outlines could communicate information more clearly to drivers than bounding boxes, bounding boxes are easier to interpret and, therefore, would be understood faster than polygons.

After evaluating the three possible tasks, it was determined that object detection would be the best for this project's AI model. The need for fast computation and existing implementations give a preference towards object detection models, which generally have less processing time compared to segmentation models and perform similar tasks for this project's needs. Furthermore, while having a precise shape of where the pothole is may be convenient for more accurate location estimation, it is not necessary, as adequate estimations can be acquired from just object detection. However, many popular object detection models for mobile deployments also come with segmentation versions as well, so if need be, there can be a switch if a more precise model is needed and can be supported by the hardware.

7.4 Model Selection

There are numerous object detection models in research and industry, some designed for real-time, edge computing tasks, and others designed to be as accurate as possible, regardless of speed. For real-time object detection, there is a major focus on minimizing model size while maintaining model accuracy.

Model accuracy is measured using mean average precision (mAP), which is the average of the intersection of a predicted bounding box and a ground truth ground box divided by the union of the area of the two bounding boxes for all bounding boxes tested against different confidence thresholds. mAP is commonly averaged using threshold confidences from 50% to 95%, which gives a reasonable estimate of how accurate the predicted bounding boxes are. Table 7.4.1 below details some of the most popular object detection models circa October 2023, with a discussion of each model and their potential use for this project. The models here have been scored on the the Common Objects in Context (COCO) dataset, which contains over 200,000 labeled images for object detection models (Microsoft, 2014).

Another important consideration is whether the models exist only as papers and GitHub repositories or are packaged for convenient use. Prepackaged models are useful because they shorten development time, as instead of needing to take time to implement, debug, and test a model implementation, an existing implementation can be fine-tuned on new data instead.

Model	mAP 50-95 (COCO)	Packaging

YOLOv8	37.3 - 53.9	Yes
YOLOv5	28.0 - 55.8	Yes
EfficientDet	34.6 - 55.1	Yes, but Deprecated
MobileNetV2	22.1	Yes, but Deprecated

FTable 7.4.1: Table of Popular Real-time Object Detection Models (Srinivasan, 2023) (Ultralytics, 2022) (Tan et al., 2019) (Sandler & Howard, 2018)

7.4a YOLOv8

YOLOv8 (Ultralytics, 2023), made and maintained by Ultralytics, is one of the most popular real-time object detection models currently. With a simple pip package and command line interface, it is easy to train and fine-tune their models with new data, with minimal setup from the developer. YOLOv8 allows for image classification, object detection, object segmentation, and object tracking, meaning it can fulfill a number of different tasks that may be useful for this project. YOLOv8 also has multiple packages available for mobile frameworks, such as Flutter, for specifically porting these models to mobile, making YOLOv8 enticing for training and exporting. However, YOLOv8's smaller models, sizes "n" and "s", have noticeable performance decreases compared to other architectures, so it might not be the best model for this task from a performance point of view. The model, however, is very fast, taking less than one millisecond to process an image.

7.4b YOLOv5

YOLOv5 (Ultralytics, 2022) is a popular iteration of the YOLO architecture that was initially released in June 2020. It had better and faster performance compared to other existing models at the time, was packaged for easy training and distribution, and had many libraries that interfaced with it directly. It has similar issues as YOLOv8, but YOLOv8 is recorded as outperforming YOLOv5 in all model sizes. YOLOv5 is another potential option, as different models, despite being older, can have superior performance in different tasks than newer ones.

7.4c EfficientDet

EfficientDet (Tan et al., 2019) is an older object detection model primarily designed towards scalability and efficiency. Unlike the YOLO models, which, while efficient, were not designed to run on extremely limited computational resources. EfficientDet was designed to run on constrained hardware resources while being more accurate than other state of the art object detection models at the time. The downside to potentially using EfficientDet is that, as the model is smaller and less complex, the model may not perform as well as the previous models. This could be a major concern, as potholes may be infrequent enough that a customer may be upset if the model misses even one obvious pothole.

7.4d MobileNetV2

MobileNetV2 (Sandler & Howard, 2018) is another object detection model oriented towards running on lower computational power devices. MobileNetV2 is an improvement over MobileNetV1 by introducing linear bottlenecks between the layers and shortcut connections between the bottlenecks, which leads to significant performance improvements over previous versions. Figure 7.4d.1 shows the overall diagram of MobileNetV2,

which is a straightforward, single pass architecture. The power of MobilNetV2 is that is it one of the fastest object detection models out there, especially for running on mobile devices. The particular version of MobileNetV2 the team investigated was MobileNetV2 SSD, where SSD stands for single shot multibox detector. As can be seen in the figure, MobileNetV2 does the feature extraction, and the SSD model performs the actual object detection in the model. For this model and the EfficientDet model, there is not any official documentation, but TensorFlow does support and Object Detection API for deploying these models on mobile devices. However, for training a model, the TensorFlow Lite Model Maker is the best tool, but the tools are deprecated and took a special configuration of Python. Training was also difficult to configure and was inefficient compared to the YOLO framework.

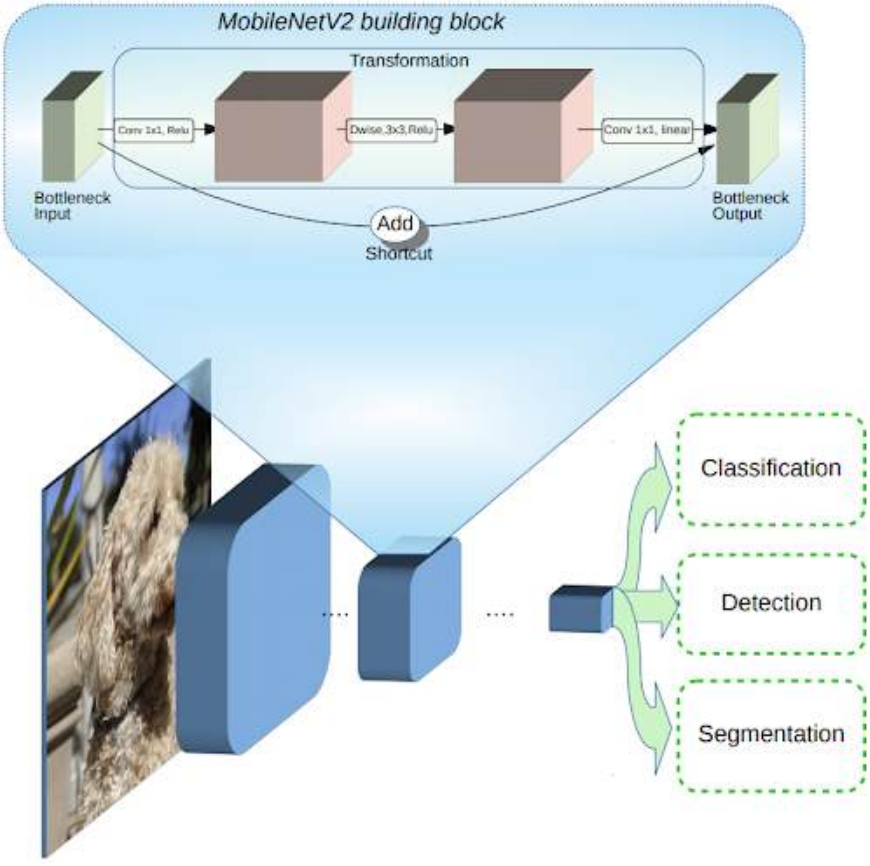


Figure 7.4d.1: Diagram of MobileNetV2 architecture.

8 Data Collection

In order to train the models for benchmarking and integration into the mobile app, they need to be trained on a custom dataset containing potholes. However, in order to assure that trained models can operate accurately and consistently in multiple types of environments, data requirements will be needed. Specifically, there needs to be quality checks that the collected data is relevant to the task, stored in a central and accessible location, and can be used for training and fine-tuning existing models to the pothole detection task.

8.1 Data Requirements

There are three major quality assurance requirements for the dataset to ensure that the model will generalize well to multiple environments and consistently detect potholes accurately. The first two are absolute requirements, and the latter are greatly encouraged, but datasets that include one of the requirements but not the other may be used.

8.1a Labeled

This requirement is simple; all the images need labels on them, and the labels need to be of potholes. The labels need to be on specifically potholes, as that is the main focus on the mobile application. While other debris may be useful later, given that it is not likely there will be many datasets that include all the extra labels, it is best to focus on just potholes specifically. Any datasets with extra classes should only be used with the pothole labeled images. Figure 8.1a.1 below shows two examples of pothole images; on the left, a pothole image without a label that would be rejected and on the right, a pothole with the required label that would be accepted into the dataset.

Figure 8.1a.1: Example Images of Unlabeled Pothole Images (Left) (Davies, 2019) and Labeled Pothole Images (Right) (Intel Unnati Training Program, 2023).

8.1b Clean

This requirement is for assuring that the pothole images do not contain any noise or obstructions that would hurt generalization during model fine-tuning. Models trained with raw noisy or shaky images will perform poorly both in training and in live testing due to poor generalization against clean data that would appear in production environments. While it is common practice to add transformations onto images in datasets to increase training set size and to improve the robustness of the object detection models, having some images already transformed while others not will create a confusing dataset of either having to transform only part of the dataset or over-transforming other parts. Figure 8.1b.1 below shows an example of a pothole image that would be rejected due to this requirement. The image is noisy and already artificially adjusted; this image would be useful if included in an already augmented dataset, but the matching untransformed image would be preferred instead.

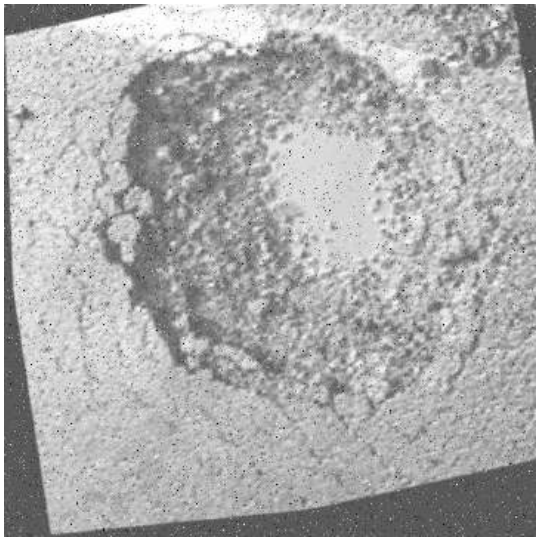


Figure 8.1b.1: Example of Rejected Image Due to Noise (Dalsaniya, 2021).

8.1c No Obstructions

This requirement encompasses the idea that the vehicle's outer windshield frame should not be obstructing the smartphone's field of view, limiting the range it can see the road and detect potholes. While a driver's setup may include their dashboard, having inconsistencies in the dataset in how much of the road is seen may worsen close detection, which could be vital in confirming the location of the potholes and notifying the user. It is also important to have a consistent view, as any blockage does not communicate anything important and instead hides information that may be important for the model's performance. Figure 8.1d.1 below shows two example images, where the left image has the dashboard covering a significant percentage of the camera's view, while the right image has minimal obstruction.

Figure 8.1d.1: Examples of Pothole Images With (Left) (Nienaber et al., 2015) and Without Obstructed View (Right) (Basily, 2020).

8.2 Found Datasets

To create an AI model that can properly predict potholes, there must be a dataset as the input. The requirements above were used to find these datasets on the internet. Two of the best dataset hosting websites are Kaggle and HuggingFace. Both sites are primarily intended for image hosting with labels, which is what is required for the AI model. These sites were the first to be scoured for the pothole datasets they provided.

Neither Kaggle nor HuggingFace had datasets that were both large enough and fulfilled the requirements that were set in place. The final dataset used was taken from three different smaller datasets and compiled together onto one core platform. Originally, there were going to be four different datasets. It was discovered that one dataset was a subset of another, so it was removed. To avoid this issue occurring with the three datasets selected, the remaining datasets were scoured for overlap. No other overlap was found. The four datasets utilized to make up the final dataset are below.

8.2a RDD2022

The first dataset found was the Multi-National Road Damage Dataset, also known as the Road Damage Dataset (RDD2022) released as part of 2022 IEEE BigData Cup under the Crowd sensing-based Road Damage Detection Challenge (CRDDC’2022) (Arya et al., 2022). The dataset includes 47,420 road images from Japan, India, the Czech Republic, Norway, the United States, and China. Of the labeled images, there are 6,544 pothole annotations. This dataset collects from a wide variety of viewpoints and locations and meets the first two requirements for the dataset. For the third and fourth requirements, different images from different countries may not pass one requirement or the other, but overall, the dataset is able to pass all

four requirements. Figure 8.2a.1 below shows some example images from China, the Czech Republic, and Japan, respectively.

Figure 8.2a.1: Example Images from RDD2022 (Arya et al., 2022).

8.2b Road Damage – Dataset Ninja

This dataset comes from Alvaro Basily collected from an unknown location. The dataset has annotations for potholes, lateral cracks, longitudinal cracks, and alligator cracks. Of the potholes annotations, the dataset has 1,331 images and 2,657 annotations (Basily, 2020). The dataset is fully annotated and has clean images, meeting the first two dataset requirements. Figure 8.2b.1 shows a collection of images from the dataset.

Figure 8.2b.1: Example Images from Road Damage Dataset (Basily, 2020).

For the third and fourth requirements, all the images are in viewpoint similar to the dashcam viewpoint, and there are minimal to no obstructions from the camera view. Therefore, this dataset was chosen to be included in the overall pothole dataset.

8.2c AI Pothole Detection Computer Vision Project

The AI Pothole Computer Vision Project dataset is a mixed dataset of potholes and other miscellaneous annotations collected from Indian roads. This dataset is a merging of 5 existing datasets, totalling to 2,218 images with pothole annotations. The dataset was created by the Intel Unnati Training Program, a program launched by Intel to help Indian students learn required skills to enter the market (Intel, 2021). Figure 8.2c.1 displays a small collection of images from the dataset. The images selected show a

common viewpoint amongst this dataset with the pothole pictures often being up close to the potholes and from the side of the road.

Figure 8.2c.1: Example Images from Intel Training Dataset (Intel Unnati Training Program, 2023).

When filtered for annotations that only include potholes, this data passes the first two core requirements. However, as can be seen, the third requirement of the dashcam point of view is not passed. However, given that two other datasets with dashcam point of views have been included, and there are no major obstructions from view in the dataset, it was decided that this dataset would be added to the overall project dataset.

8.2d Road Pothole Images for Pothole detection

This dataset is a pothole-specific dataset released in 2018 collected near Stellenbosch, South Africa. This dataset is different from the others in that it is split into two sub-datasets, a simple and complex dataset with some overlap. Overall, the dataset has 2,008 images with 7,588 pothole annotations between the two sub-datasets (Nienaber et al., 2015). Figure 8.2d.1 displays a collection of images from the dataset. As part of processing, the two sub-datasets were merged and duplicates removed, so images displayed here will not be indicated from which sub-dataset they originate from.

Figure 8.2d.1: Example Images from Stellenbosch dataset (Nienaber et al., 2015)

For the dataset requirements, after filtering the images with pothole annotations are kept and all the images are clean with no noise. For the supplementary requirements, as can be seen in the figure above, the field of view is in the dashcam position. However, the dashboard is significantly obstructing the view of the camera, so the fourth requirement is not being passed. However, given the balance of viewpoints from the other datasets and the complex scenarios represented in this dataset, it was decided to add this dataset to the overall pothole dataset.

9 Dataset Hosting

In order to host all the data collected, a robust cloud storage provider needed to be found. After looking at the available storage options, including OneDrive and Supabase, Roboflow was chosen as the primary site to host the dataset. There are three reasons for that:

- Roboflow provides ample resources and video tutorials on running AI models with their technology, specifically including YOLOv8 and YOLOv5, which are on the list as potential models of choice.
- Roboflow's uploading dataset procedures allows the app to upload many different types of annotations on images from many different websites. Roboflow then will sort through them all, upload, and combine the image and annotations automatically.
- Roboflow provides many different resources for manipulating the dataset. This includes removing unwanted classes, editing existing annotations, performing augmentations to the dataset, and configuring transformation to run when downloading the dataset for training.

For the app's purposes, Roboflow is being used more for its dataset abilities, despite having cloud-based models available. Roboflow provides the ability to test an AI model on YOLOv8 and YOLOv5 using Colab, using a straightforward process with many resources. For the final model that is trained, Roboflow will not be used, as the app requires the ability for the model to work offline. Roboflow does not provide that capability without direct integration and a price increase. The final dataset link can be found at https://universe.roboflow.com/pothole-wipm4/ai-pothole-detection-hitzl.

9.1 Dataset Divisions

During the process of uploading the datasets to Roboflow, they are automatically divided into three separate sub-datasets: the Training, Validation, and Testing sets.

The Training set consists of 70% of the images in the entire dataset. These are the images that will be primarily used during model training and fine-tuning and will be used to teach the model how to detect potholes while on the road. This dataset, as the largest, is expected to be well-balanced

and have a representative sample of the different locations and viewpoints previously discussed in the uploaded datasets.

The Validation set is the following 20% of the dataset used for benchmarking the model performance during training and to give a performance benchmark during any hyperparameter tuning. The validation dataset is used during training to test if the model is potentially overfitting on the data. If the training performance is still improving but has stalled on the validation data, then the model is overfitting and should be stopped early or modified to prevent overfitting.

The testing set is the final 10% of the image dataset, and is used to judge the final performance of the model. The images in the testing dataset are meant to never be seen by the model during training or validation, and are a final benchmark to compare model performances.

9.2 Dataset Uploading

The first dataset uploaded was the RDD2022 dataset. This dataset needed to be manually divided and filtered in order to remove non-pothole annotations, as some locations possessed more than the 10,000 image limit set by Roboflow for their free tier. This dataset contained 2,300 images that were relevant to the project with the 6,544 annotations.

The second dataset that was uploaded was Dataset Ninja. Dataset Ninja had four separate labels, but the only important label for the app to use is the pothole labels. Removing them was difficult. The dataset was uploaded to a separate Roboflow model, then pre-prepared for running a model. Pre-preparing a model is what allows the dataset to get sorted into Test, Train, and Valid. By doing this, the model was set to only look for pothole images. This still left in all of the images that have different labels in

the dataset. It only means that they have no potholes in them. The next task that was taken was to remove all images that have a “null” amount of labels in the dataset. This left only the images that have potholes in them and only potholes. 1,000 images from this dataset were used.

The third dataset that was uploaded was the Roboflow dataset with dashcams. This dataset was pulled from the same website, Roboflow, that the images were hosted on, so it’s reasonable to assume this process is easy. However, there were some difficulties. To upload the dataset, a new model was branched from the dataset, and the process to train the model was begun. Only then was the dataset able to be downloaded. The dataset was then uploaded for 1,500 images.

The final dataset uploaded was the Stellenbosch dataset. Before uploading to Roboflow, the simple and complex sub-datasets had to be merged. This was done by moving all the images in the simple sub-dataset with potholes annotations into an output folder and logging their hashes, then moving over the complex sub-dataset into the same folder, ignoring any images that had hashes that were previously seen. In total 9,303 images were uploaded.

10 Model Training

There are a wide number of models that the app could potentially use. The original intention was for the app to use YOLOv8, but there are a large number of options that the final dataset can be tested on. Before implementing the model into the mobile network, different models were tested on different datasets. First, the models were tested on a rough-draft Pothole dataset taken from HuggingFace to serve as a test of the different models while still in the pre-processing phase. This test was used to assess

the feasibility of the project as a whole. If an AI model could not identify potholes in a useful way with high enough accuracy, then the project cannot work with this method. The dataset was tested with some code written in Python on a group members' Macbook. These early runs were luckily a success. There was potential in the implementation of AI models on pothole datasets.

Once the final dataset was compiled into Roboflow, where the data was kept (see AI Database Storage), the dataset was tested on YOLOv5 and YOLOv8 to compare their differences in performance across different metrics. YOLOv8 is the newer model with features that YOLOv5 doesn't have, whereas YOLOv5 has a more streamlined implementation with Flutter; there is a direct and tested Flutter third party software which implements it straight into the app. The team also trained a MobileNetV2 model after testing the YOLOv8 and YOLOv5 on a mobile device and not being able to reach real-time performance.

10.1 Training Setup

The YOLOv8 and MobileNet models were trained using one of the team member's RTX 3090 GPU, as it trained faster than using UCF's Newton server and using the GPUs available on Google Colab, while the YOLOv5 model was originally trained using Google Colab.

YOLOv5 was trained for 50 epochs with a learning rate of 0.01 and YOLOv8 was trained for 100 epochs with a learning rate of 0.01. Roboflow provides outline code for starting to test with YOLOv8 and YOLOv5, which was used. The MobileNet model was trained for 70,000 steps, in which a single step is a single batch of images, using the TensorFlow Object Detection API. However, the output is the completed AI model and that can be installed with Flutter. The model is simply being trained differently. The

Colab that runs the AI model for YOLOv5 does five things to prepare the model to be run:

- Implements OS to use. YOLOv8 specifically is primarily intended to run using OS, so this is a necessary step.
- Installs the corresponding model from their own website (YOLOv8, YOLOv5)
- Uploads Roboflow's own resources for uploading a dataset into the local machine
- Uploads the dataset from Roboflow's systems into the model itself. This process requires a key, which is provided to the owner of the dataset.

- Custom trains the model on the GPU associated.

    Colab, with a premium plan,provides a T4 GPU on the proper systems. This is a useful boon to running code while also being able to adapt the code on the fly. Colab's systems are optimized for preloading different code for quick manipulation of specific parts of code. Once these systems are in place, and a few smaller systems, the model can be run. The process for running YOLOv5 and YOLOv8 are slightly different, but both use the same outline as detailed above.

In order to evaluate the performance of the models, the following metrics were used:

- mAP- 50: The Mean Average Precision (MAP) is an evaluation metric in object detection. It measures the average precision of the AI model over a certain threshold – in this case, the threshold 50% (50% confidence. It measures how good the overlap between the data in the dataset and the model's prediction is.

- mAP 50-95: An extension of the MAP metric that extends from 50% to 95% confidence, instead of just 50%. It gives a more comprehensive view of model performance over broader thresholds.
- Recall: The average of the number of bounding boxes are correctly drawn in a location versus the total number of bounding boxes in the dataset itself.
- Precision: The average number of times the predicted bounding boxes have the correct class and location to the ground truth label.

10.2 Results

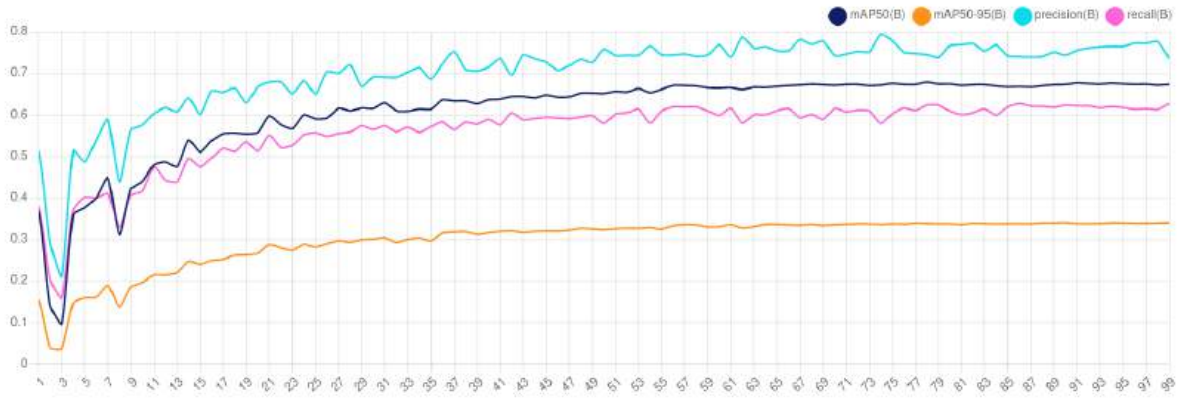
After training both models, YOLOv8 had a clear advantage in performance over YOLOv5. In mAP-50, YOLOv8 outperformed YOLOv5 by 19.8%, and outperformed by 15.9% in maP 50-95. In terms of recall and precision, YOLOv8 again outperformed with 16.8% and 10.1%, respectively. Figure 10.2.1 displays the performance results of YOLOv8 and YOLOv5, and the following Figures 10.2.2 and 10.2.3 display the training performance results on the validation dataset.

Model	Recall	Precision	mAP 50	mAP 50-95
YOLOv8	0.614	0.778	0.673	0.34
YOLOv5	0.46	0.610	0.475	0.181

Figure 10.2.1: Results Table after Training on Pothole Dataset

YOLOv8's higher mean average precision means that the model will most accurately detect potholes and will more accurately detect their true location. Furthermore, YOLov8 better recall and precision metrics give increased confidence that the model will not both detect potholes that don't exist nor miss potholes that appear in the images. Given YOLOv8's clear

performance difference on all metrics compared to YOLOv5 and the ease of implementation for both models, YOLOv8 is the current model selected for integration with the laptop application. For the mobile application, we went with the MobileNetV2 model the team trained, as it had the lowest latency on the testing device, which was a Samsung Galaxy S10+ running Android. The latencies can be found on Figure 10.2.4. The team deployed all the models after running through their respective mobile optimization pipelines and performed INT8 quantization. All models had a notable drop in performance after quantization, but the MobileNet model was able to meet the speed requirements once quantization was applied. However, as will be seen later, the quantization hurt the mobile performance too much to be usable, but as it was the only model that could be ran on the mobile device, the team switched to developing a laptop program instead. For the laptop program, the team used the original YOLOv8 model we trained, as the laptop the team was using had a GPU that could easily run the model in real time.



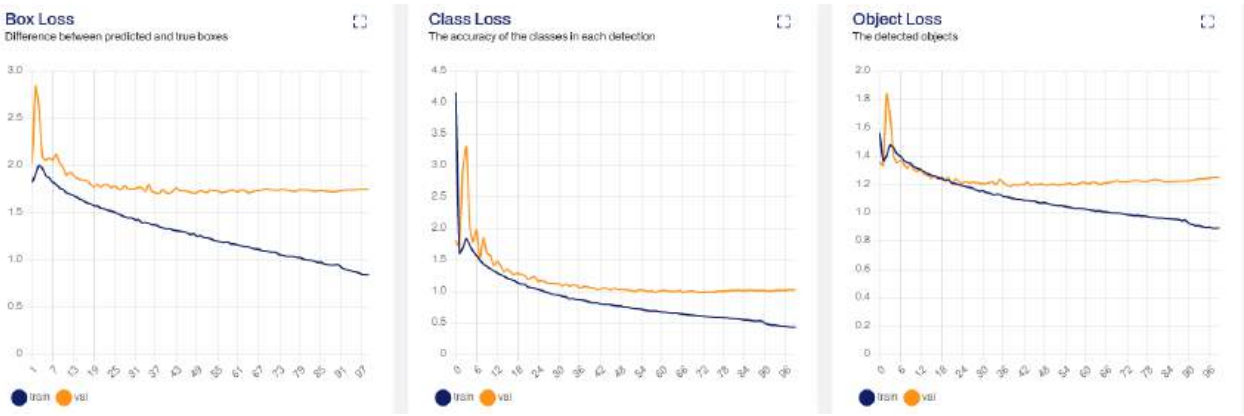


Figure 10.2.2: Per-Epoch Validation Performance for YOLOv8

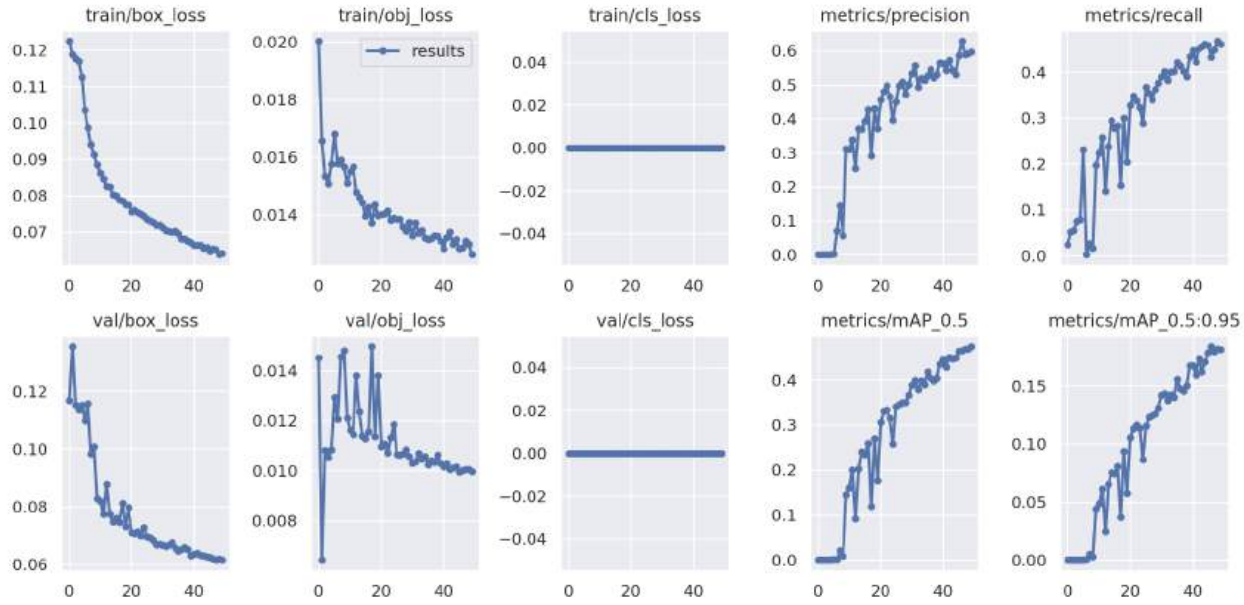


Figure 10.2.3: Per-Epoch Validation Performance for YOLOv5

Model	Latency (ms)
YOLOv8	880
YOLOv5	650
EfficientDet D0	350

Real-time Requirement (5 FPS)	200
MobileNetV2 SSD	150

Figure 10.2.4: Per-Epoch Validation Performance for YOLOv5

11 Mobile Design and User Experience

11.1 Application Pages

The application has three main pages and four pages overall. It is strictly in landscape mode because that allows the camera feed to have the best field of view for the model to process and interpret. The application also has a swipeable app drawer with buttons to allow the user to navigate between the pages.

The first page is a welcome screen which will only be forcefully shown to the user on the first time they open the application. This page provides the user with a warm welcome to the application and gives a brief tutorial of what the application does and rules on how to use it effectively. Once this page is dismissed it will not be shown in the navigation app drawer but it can be viewed again at any point by accessing it from the settings page.

The second page is the main view of the application and it is a camera view. This application essentially turns the user's smartphone into an interactive driving aid so it is important for user experience to display the camera feed of what the phone camera and model sees while the user is driving. Not to compare it to another application but it is very similar to the SnapChat user interface in which the main application page is the phone's live camera feed. This page displays nothing other than the camera feed, aside from road obstruction alert interface elements but this will be explained later in this document.

The third page is the settings page. While this page sounds boring and uneventful, it is the most important page for the user and for the app's functionality. The settings page consists of multiple tabs, buttons, toggle switches and sliders that control all of the application features and functions. There is a button that leads to a "More Information" page which displays the first page discussed in this document. There is a button that resets all settings back to its initial settings. There are toggle switches that handle alert settings and notification settings that the user may or may not want. Since the group has a visual and audio alert when a pothole is detected, users have the option to turn those off if they are uncomfortable with either setting.

The fourth page is the map page. This page is a stretch goal that allows the users to view all the locally reported potholes in their area on a map. The map service used is Google Maps, as it was agreed by the team since the group thought it has the smoothest experience, pleasant aesthetics, and easiest implementation for flutter. When the user first opens the page, the map is taken directly to the user's location. The potholes are displayed on the map with Google's pin markers.

These four pages have the following navigation flow. When a user first opens the application initially, it displays the welcome page. From there, they navigate to the App Drawer which houses the Camera and settings view. From there they may navigate to either the camera view or the settings view with the app drawer. From the camera view, the user may navigate to the settings view and vice versa with the app drawer. The map button in the app drawer navigates them to a separate page for the map view, and can navigate back with the system's back feature. And from the settings view, the user may navigate to the "more information" page to access more information about the app's functionality if they ever forget. Below, Figure 11.1A is a diagram of how the app's navigation flow is

demonstrated.

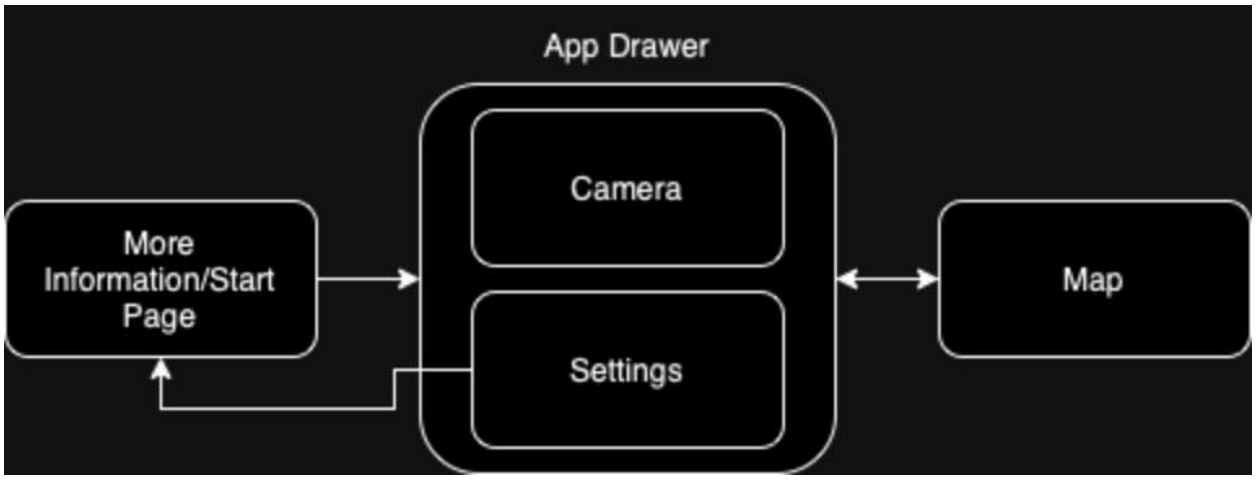


Figure 11.1A: Git Process

11.2 Figma

In order to start planning for mobile development the team has created a Figma for the app. This allowed the team to start picturing what the application will look like and make any easy changes to the UI before the group starts actually developing the app. The Figma was made by only 2 members of the team but they have taken inputs from everyone involved in the project in order to try and keep everyone up to date and implement any good ideas that come up.

All of the screens were subject to change based on time constraints. The team made the application look as professional and user friendly as possible, but first, the team had to implement the model into the app. The work on the application started before the AI models were trained, so while waiting for the models, the user interface was polished before the AI model was implemented. The group spent time afterwards in making sure the application is fully functional.

Aaron built the skeleton of all of the screens and Andy, in mobile development, took the time to make it look better and added in some pictures.

11.3 Camera Screen

Figure 11.3A is the “Camera Screen” which is the most used screen and the default screen when running the app. The screen is strictly in landscape mode because of the fact that the model is trained in landscape mode. There might be issues with the model if it is trained in landscape and then tested in portrait, so the team plans on limiting the application to only be in landscape for now. One of the goals this group has is to manipulate the camera whether it is filming in landscape or portrait in order to allow the user to mount the camera as they like and remove the restriction of landscape only in the app, therefore bettering the user experience.

The red bars on the sides of the screen, as shown in figures 11.3B and 11.3C, appear when the model detects a pothole and the user is going to hit it. If the model detects the pothole on the left side of the road it gives a visual notification with the left bar extending out onto the “Camera Screen'' along with the caution symbol.

When the banners are shown independently, a sound is played for each side. When a pothole is detected on the left side, the sound that is played is a voice that says “left”, and on the right side, the sound that is played is the same voice that says “right”. When the setting is disabled and the banners show on both sides at the same time, one sound is played which is a more general alert tone.

It is understood that if the driver of the vehicle is the only one in the car while using the app, then the visual cue can be distracting and unsafe. In

order to account for that, there are two different audio cues that play depending on which side of the road the pothole is in order for them to avoid it and not be distracted visually. The team took steps to allow the user to change the opacity of the red bars in order for it to not be as distracting as well as change the volume of the audio cues as needed.

Figure 11.3D shows a pothole being detected with the setting for detecting potholes on the left side and right side independently disabled. It also shows how the bounding box looks like around a pothole. This is useful for the user to see physically where the pothole will be on the road, especially if this option to show the banners independently is disabled.

When the camera page is first opened, a road overlay appears on the screen to help guide the user's camera view line up directly with the road ahead of them. This is shown in figure 11.3A. This is important because the application is designed to detect potholes in the general area where the driver would usually have the road showing on screen.

Figure 11.3a Road Overlay

Figure 11.3b Left Warning Banner

Figure 11.3c Right Warning Banner

Figure 11.3d Bounding Box

11.4 More Information Screen

Figure 11.4A is the application's "More Information Screen." This screen opens the first time the application is opened to allow the user to understand what the application does better and how they have to set up their phone for it to work as intended. In this particular rendering, the team decided to make a significant modification by altering the original font choice that the team had initially selected. The reasoning behind this decision was to transition from the initial font, which may have conveyed a more casual or informal tone, to a font with a formal and serious aesthetic. This transformation was intended to instill a sense of authority and promote a heightened level of user confidence in the application.

As it was stated above, the model was trained in only landscaped pictures and videos so the application itself must be in landscape. The phone will have to be mounted on the dashboard of the car with the camera facing towards the road, out the windshield. This gives the best point of view of the road along with having a wide point of view that makes it a lot easier to calculate whether the pothole is appearing on the left side of the road versus the right side. Having this wider view will also tremendously help when trying to calculate if the user will actually strike the pothole or not. When filming in portrait the camera is unable to obtain as much information and therefore it might not be as accurate when making these calculations.

Figure 11.4a: More Information Screen

11.5 App Drawer

The main app runs on a background page which is actually a drawer. This drawer can be accessed by swiping left to right. As seen in figure 11.5a, this reveals a section that shows the title of the app, and buttons that navigate to the camera page, settings page, or the map page. The team chose to utilize semi transparent backgrounds for the buttons located at the bottom of the drawer, which house essential settings and camera options. This design choice was driven by the goal of creating a more streamlined and less invasive interface, ultimately reducing the visual distractions that users may encounter during their interaction with the app.

The use of semi transparent backgrounds serves a dual purpose. First, it allows users to focus on the app's primary content without overwhelming them with overly prominent buttons. By softening the visual impact of these buttons, it allows the application to maintain a clean and uncluttered user interface, making it easier for users to engage with the app's core functionalities. This approach aligns with principles of minimalist design, prioritizing content and user interaction over unnecessary elements.

In addition to selecting appropriate icons, the team paid close attention to the placement and arrangement of these buttons. Their positioning was carefully considered to ensure that they contribute to the

app's overall aesthetic. By aligning them evenly and integrating them into the app's layout, aiming to create a refined and modern look. This not only improves the visual appeal of the app but also enhances the overall user experience by providing an interface that feels cohesive and visually pleasing.

To further improve the accessibility and user-friendliness of the app, the mobile team opted to use icons within these buttons. Icons serve as a visual reference that transcends language barriers, making it easier for users to quickly identify and understand the purpose of each button. This not only enhances accessibility for a broader user base but also simplifies the user's learning curve, as they can readily associate visual cues with specific actions or settings.

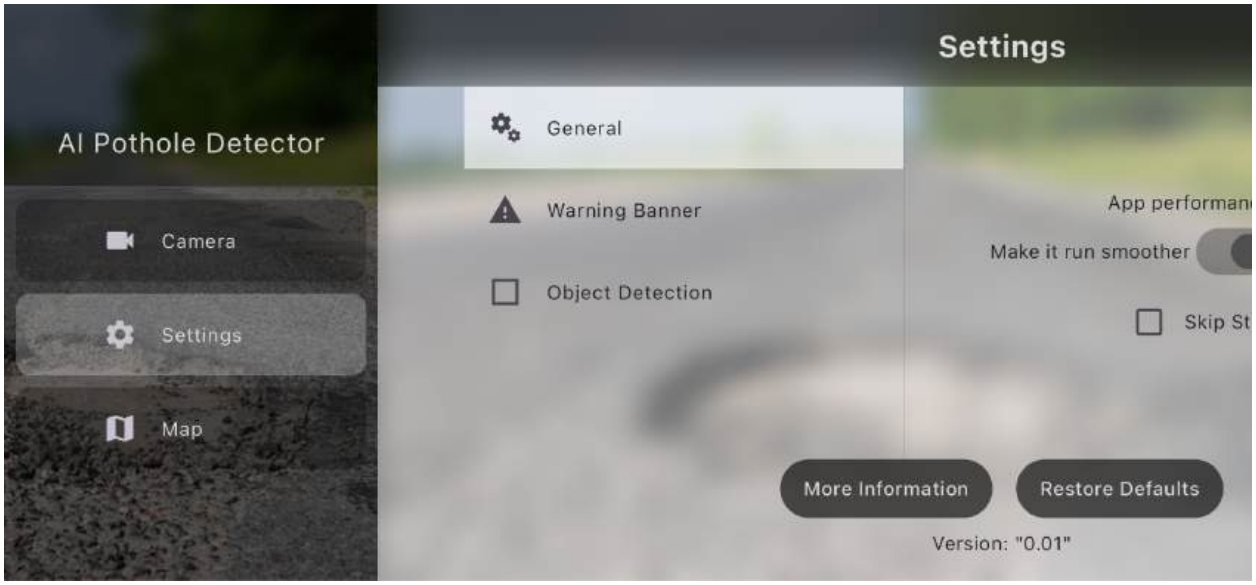


Figure 11.5a: App Drawer

11.6 Settings Screen

This is the final screen that is important to the application. This screen is crucial in allowing the user to change settings to their liking as well as learn any information they need about the application. In designing this

screen, the mobile team aimed to maintain a coherent and visually pleasing user interface that aligns with the design language of the entire application.

There are a few notable design choices in this screen. One notable design choice was to maintain the semi-transparency in the background. This decision was made to ensure that the aesthetics of the settings screen remain consistent with the overall visual theme of the application. By keeping this design element, it creates a sense of continuity and unity in the user experience. The other notable design choice was keeping the format of choosing different buttons to show different information as tabs, similar to how the app drawer is designed. These design choices provide users with a familiar and cohesive environment as they navigate through different sections of the app. This consistency is important because it reduces cognitive load, making it easier for users to understand and interact with the settings screen seamlessly.

The following figures show the user interface for the settings page. At the bottom of the page, two buttons are shown. One is the more information button, which takes the user back to the same message that shows when the app is first opened. The other button is the “restore defaults” button, which when tapped, a confirmation dialogue appears that asks the user for confirmation to reset all settings back to the initial default settings.

In figure 11.6a, the “General” tab is open and shows the settings for making the app have a more aesthetically pleasing look, or to have it perform better. It also contains a check box that gives the user the option to skip the “more information” screen when the app starts or to bring it back if the user previously checked the “never show again” box.

Figure 11.6b shows the “Warning Banner” tab open in the settings page. This section shows the settings that deal with the warning banners on the camera page. This is where the option to show the left and right

banners independently is set. Other options are available to customize the look of the banners, such as background blur, opacity, and the alert volume. The "Alert Volume" and "Alert Opacity" sliders are especially noteworthy for their flexibility. There is the slider to increase or decrease the volume of the notification that plays. Alert Opacity allows the user to increase or decrease the opacity of the notification bars that are visually displayed on the screen while the application is in use. This is very important as if the bars are too distracting for the driver, they are able to turn down the opacity or turn it all the way off entirely.

Figure 11.6c shows the “Object Detection” tab open. This section shows the settings that deal with the detection of potholes.

- The option to show latency info is mainly used for debugging. As shown in the image to the right, all info related to time for converting camera stream images, processing time, camera stream resolution, and frame rate is in red.
- The option to show boxes is what allows the users to see and visualize where the pothole will be in addition to the alert sound that plays when a pothole is detected.
- The option to show classification text is to show the text and confidence level inside each bounding box around the potholes.
- The confidence threshold slider allows the user to control when the application shows the detected potholes that have a confidence level above the confidence threshold. When lower, the application is more likely to show detections that are not actually potholes (false

positives). When higher, the application may not show all potholes on the road (false negatives).

- The camera resolution slider allows the user to control the resolution of the camera stream in case their device cannot handle higher resolutions.

The inclusion of toggles and sliders is another significant design choice. These interactive elements provide users with the means to tailor their experience within the app to their precise preferences. The choice to offer a slider with no indents or specific predefined selections within the slider reflects a commitment to user customization. It allows users to finely tune their settings to meet their unique needs and preferences. This level of granularity empowers users, making them feel more in control of their experience, and it's particularly beneficial when preferences can vary widely from one user to another.

More Information opens the “More Information Screen” that was just talked about above allowing the user to read about the application in case they forgot the requirements to have the model and application work as intended or on the off chance they skip the screen on the first open and need to have it opened again. The other button is the “restore defaults” button, which when tapped, a confirmation dialogue appears that asks the user for confirmation to reset all settings back to the initial default settings.

Finally, the Version of the application is placed in the Settings Screen for when there are bugs the user is able to check and see if they are currently running the application with the latest version or if they need to update it. The Navigation Bar is still displayed at the bottom of the screen for easy navigation to the other screens.

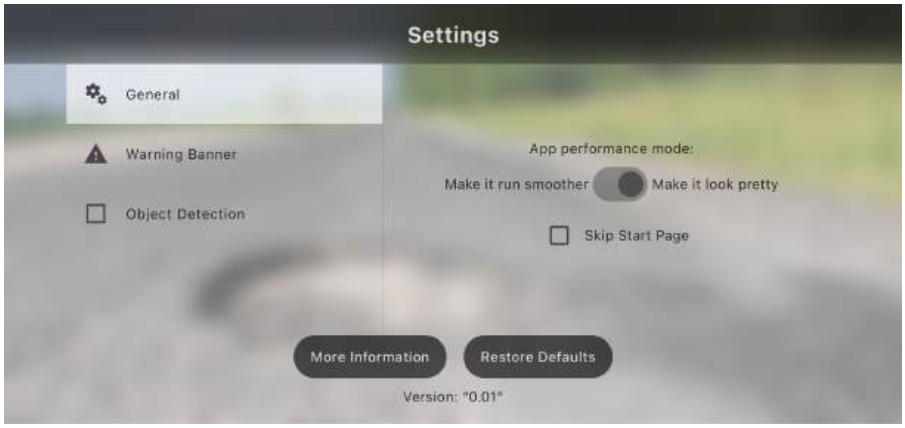


Figure 11.6a: General Tab in Settings Screen

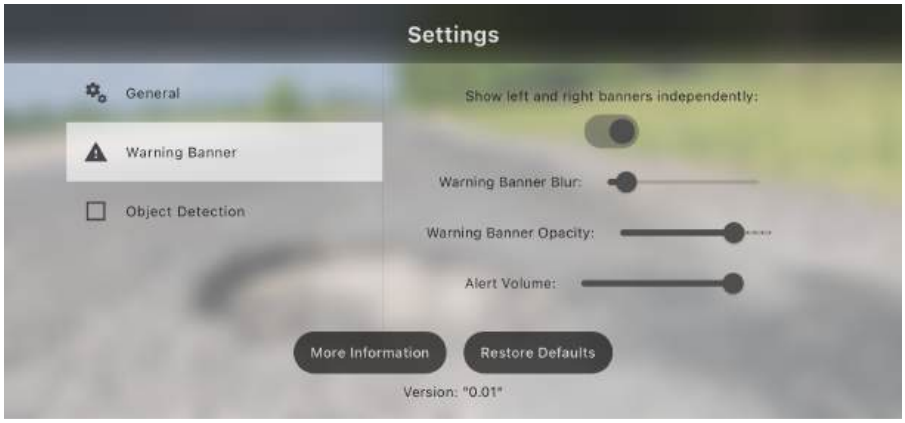


Figure 11.6b: Warning Banner Tab in Settings Screen

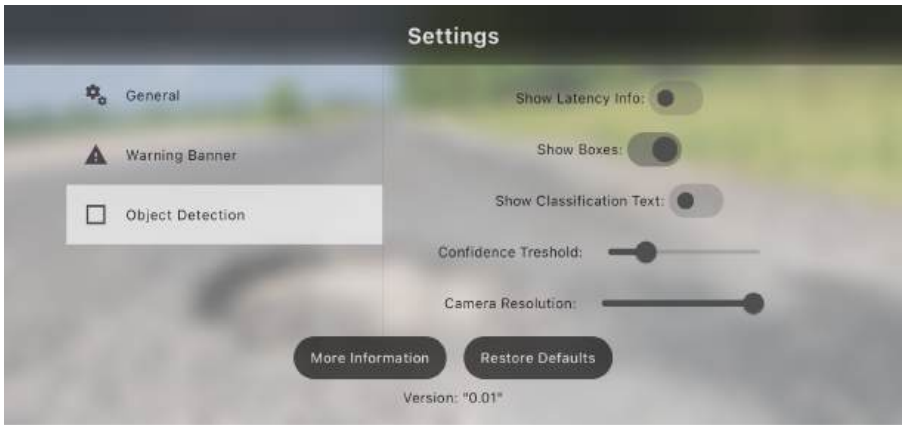


Figure 11.5a: Object Detection Tab in Settings Screen

11.7 Mapping in Mobile Application

One of the main stretch goals of this team is to store the locations of all of the potholes this app has detected in order to better predict when potholes are coming up. Having this implemented allows the application to warn the user of potholes very much in advance. There is no screen made for this in the Figma because it is a stretch goal and the group primarily focused on implementing the user interface and getting a working model implemented into the mobile application first.

The group was able to create a fully fledged application with time to spareto implement this functionality as the first priority. More about this concept is described later in the paper. When it comes to User Interface and User Experience, the mobile team has added a third main screen that allows the user to see a map with different potholes that have been detected. This is a very simple screen similar to any mapping application that already exists with warning symbols at the location of the potholes that the group has found.

Like it has been mentioned it is very important to the user experience because it can help warn the driver and help prevent an accident. Say for some reason that a user’s phone is unable to warn the driver in time because of processing issues, the phone is old and the application is unable to work in the amount of time needed, or any other issue that occurs, the user will still be able to receive a warning based on the fact that the camera and model will not need to detect.

Figure 11.7a: Mapping

11.8 Libraries Used

shared_preferences: ^2.2.2

This library is used for settings across the whole application.

camera: ^0.10.5+6

This library is used for the camera stream to detect potholes on the road.

native_device_orientation: ^2.0.3

This library was used to be able to keep the camera stream orientation lock consistent with the rest of the application's landscape orientation lock.

audioplayers: ^5.2.1

This library was used to have the ability to play sounds when a pothole is detected.

tflite_flutter: ^0.10.4

This library is used to make use of the TensorFlow Lite AI model that was trained for this application to detect potholes.

image: ^4.1.7
This library was used for the ability to manipulate the images that are taken directly from the camera stream to process for pothole detection.

google_maps_flutter: ^2.5.3
This library is used for the user to be able to navigate the maps using Google Maps to discover potholes locally around their current location.

geolocator: ^11.0.0
This library is used to get the user's current location, mainly used for storing pothole locations in the database.

permission_handler: ^11.3.0
This library is used to get permission from the user to grant device's access to camera and location.

http: ^1.2.0
This library is used for the ability to call API endpoints in the application.

12 React Native vs Flutter

12.1 Flutter, what is it?

The team has looked into the possibility of utilizing Flutter for mobile app development over React-Native. Flutter, a Google open-source UI software development toolkit, has expanded significantly since its conception, redefining cross-platform app development. Flutter has experienced tremendous changes as a result of Google's goal to address the issues of developing high-quality, visually appealing, and performant mobile

applications. It has cemented its position as a powerful framework for developers globally.

12.1a Early Development and Origins

Flutter was first announced by Google at the Dart Developer Summit in 2015 as a project called "Sky." Its primary goal was to provide a rendering engine that could handle a variety of platforms, including mobile, desktop, and web, by utilizing the Dart programming language. The project's goal was to address the limits and complications that developers encountered when developing cross-platform programs.

Flutter's first alpha version was revealed by Google in 2017, marking the start of the framework's journey to becoming a renowned platform for mobile app development. Flutter received significant feedback from developers during its alpha and beta phases, resulting in iterative upgrades and enhancements.

12.1b Key achievements and enhancements

With the release of version 1.0 in December 2018, Flutter reached a critical milestone. This stable version marked Flutter's ready for production use, indicating the framework's maturity as a framework capable of developing high-quality apps for iOS, Android, and even the web.

Flutter's primary strength is its outstanding performance. Google has consistently worked to improve Flutter's performance by increasing rendering rates, decreasing app launch times, and optimizing the framework's efficiency, resulting in smooth and responsive user experiences.

Google has continually added new features and advancements to Flutter, hence boosting its potential. Rich set of UI widgets, hot reload for rapid development, and Flutter Inspector for debugging have all become essential components for developers. Furthermore, advancements in tools and interaction with IDEs such as Android Studio and Visual Studio Code improved the development experience.

Flutter's flexibility outside mobile platforms have been a primary goal. Google added web development support to Flutter, allowing developers to compile Flutter code to run natively in web browsers. This enhancement boosted Flutter's appeal as a comprehensive cross-platform development solution.

The Flutter community has seen exponential expansion, with libraries, packages, and resources contributing to the ecosystem. Google's extensive community engagement, which included events like Flutter Engage and projects like Flutter Create, encouraged innovation and cooperation.

Dart, Flutter's primary language, was also improved. Dart grew to become more robust, with new features, increased performance, and improved tools, complimenting the development ecosystem of Flutter.

12.1c Recent Developments and Future Prospects

Flutter has recently provided enhancements such as better support for desktop platforms such as Windows, macOS, and Linux, making it a more comprehensive framework for multi-platform development.

For example, Dart 3 is a wonderful upgrade to Flutter 3.10's most popular programming language. The elimination of non-null-safe code in

Dart 3 ensures a completely secure and error-free experience in this 100% safe language and eliminates the common hazards of nullable languages.

Dart 3 also includes various language enhancements, such as the introduction of Patterns. Because of this capacity, working with structured data is simple. The official Dart 3 blog post contains a practical demonstration of a function that conveniently returns two values at once.

As a result, there is no need to construct a separate class for that purpose or to encapsulate several values within a collection.

Furthermore, new class modifiers like interface class and sealed class provide enhanced possibilities, and the modified switch statement allows for the systematic breakdown of structured patterns.

The application loading time of Flutter for the web has been significantly improved.

CanvasKit, the largest Flutter for the web component, has been reduced in size to one-third of its previous size, which is a significant feat.

Furthermore, unwanted typefaces can now be removed, lowering overall weight. Flutter 3.10 now has full support for quickly integrating pure HTML components throughout the application.

Furthermore, with the addition of fragment shader capability, developers may now use well-known Dart code to build spectacular visual effects.

This means including garbage-collected languages like Flutter in the standard. Initial testing has revealed a three-fold increase in performance. When WASM is released to the public, this exciting development holds immense promise for web applications built using Flutter.

Flutter can be used for web development in a variety of ways, from installation to creating layouts and animations.

12.1d New Material 3 Widgets for Flutter for Web

Flutter 3.10, the most recent version, now has improved support for Material 3. It enables you to generate color schemes using a base color or an image. There are also substantial enhancements in different widgets like DropdownMenu, NavigationDrawer, TabBar, SnackBar, and many more in Material Design components.

12.1e Drawer for Navigation (Navigation Drawer)

In addition, the Flutter team has improved support for iOS/macOS. This enables users to use Apple's Spell-checking capability within editable text widgets, as well as a new checkbox and radio button design that matches the Cupertino aesthetics. There are also several animation enhancements tailored to Apple platforms.

In terms of Apple devices, this Flutter version now supports wireless debugging directly on iPhones and iPads. This functionality, however, was previously exclusively available in Xcode.

12.1f Improved DevTools

The development tools have also been improved in this Flutter 3.10 release, allowing developers to more rapidly examine and enhance the performance of their apps.

New features have also been added to the memory page. The addition of the Diff tool allows you to compare memory utilization before and after

specific interactions to evaluate the effects of those interactions. Additionally, heap exploration via the console has been optimized with improvements.

The DevTools user interface has been enhanced with the addition of Material 3 widgets, enhancing usability and complying to current design rules.

Perfetto, an open-source program, has also replaced the obsolete trace viewer. Perfetto excels in managing large datasets, in addition to introducing features such as pinning threads of interest, dragging and selecting multiple timeline events, and using SQL queries to retrieve specific timeline data.

Impeller for iOS that is ready for production

Skia has been replaced with Impeller with the release of Flutter 3.10. It has surpassed OpenGL as the primary rendering engine for iOS. This new rendering engine improves animation performance and eliminates the vexing shader compilation difficulties that produced janky animations and a poor visual experience.

To achieve this result, Impeller employs a tessellation technique, which eliminates the requirement for shared compilation during graphics rendering.

Impeller takes advantage of the cutting-edge capabilities of next-generation GPUs to smoothly generate diverse forms and colors on the screen while maintaining a high frame rate. It has been deliberately designed from the ground up to specifically meet Flutter's demands.

Impeller is the default rendering engine in all iOS apps produced with Flutter 3.10. According to the Flutter team, a preview version of Impeller for Android is also planned for future releases, and they have also highlighted their ongoing efforts in this respect.

12.1g Utilizing Dart

Flutter is a framework that utilizes Dart, therefore what is Dart? Members of Chrome's V8 Javascript engine team made Dart because they were fed up with some parts of the 20-year-old language they had to use every day. The group just got back from a Dart Developer Summit where they showed off the Dart on Android project. Instead of a clear name like "Dart on Android," Dart on Android is called "Sky." Right now, Sky (Dart on Android) is just an open-source test, but the project has a lot of potential compared to the usual way of making apps.

12.1h Flutter Interact

Flutter 1.12, the newest stable release of the Flutter framework, was just announced at Flutter Interact. The most recent quarterly release is the result of the efforts of hundreds of people both inside and outside of Google. It includes additional speed enhancements, increased flexibility over integrating Flutter content to existing apps, and Material and Cupertino library upgrades. They also offer a new Google Fonts package that gives you direct access to almost 1,000 open source font families, putting stunning typography at one’s fingertips with just a single line of code.

12.2 Why Flutter over React-Native

Flutter, being a cross-platform solution, allows developers to create apps for both operating systems using the same codebase. However, the user claims that this is not the only reason why adopting Flutter makes the development process faster and more efficient. It also has hot restart and the well-known "hot reload" option, which allows the user to see updates in real time without having to restart the program.

This considerably accelerates the development process. React-Native and Xamarin provide comparable functions, however they are slower. Flutter's characteristics help users to save time and resources.

12.2a Widgets and Compatibility

Widgets are the building blocks upon which the whole program is built. There are ready-made as well as customizable ones - anything in Flutter may be constructed from widgets, according to the user. Because widgets are part of the app rather than the platform, the finished product will most likely have less compatibility concerns across platforms and OS versions.

Individuals who continually strive for greatness and push themselves to realize their full potential achieve high performance.

According to the user, Flutter applications execute at a level equivalent to native mobile apps and outperform alternative cross-platform solutions. This is mostly because Flutter is the only mobile SDK that does not need a bridge (JavaScript or webview) to communicate between the app and the platform. As a consequence, the user benefits from a faster-starting software with gorgeous, rapid animations and fewer performance issues.

12.2b Mobility

Many developers are already creating Flutter products that operate on the web, desktop, and even TV. The features described are still in various phases of development, but Google is more than capable of producing a stable release. Given how essential IoT is to the firm, it is quite probable that Flutter will continue to evolve in this way.

Internationalization and accessibility are critical issues in today's digital environment. They guarantee that products and services can be used and

understood by people from many nations and with varying abilities.
Internationalization

Internationalization, or the creation of multiple language and area versions of an app, often occurs after the program has been established and can result in a variety of inconsistencies. Flutter provides tools that make the process simple and integrate it directly into development. Flutter also supports accessibility by offering bigger fonts, screen readers, and better contrast, all of which are automated from inside the platform.

12.2c Open Sourced

Open source and an engaged community are essential variables in software development. They enable cooperation and input from a diverse group of people. The open-source aspect of a project means that the source code is freely available for anyone to read, edit, and share. This promotes innovation and motivates developers.

Flutter, as an open-source platform, is free to use and has a growing community that contributes to its excellent documentation and aids developers with difficulties they may experience. There are also numerous YouTube tutorials accessible for anyone who wishes to study Flutter or develop their abilities in Google's mobile UI framework.

12.2d Shorter Time to Market

One of the most compelling reasons to choose Flutter development is its inherent potential to substantially accelerate the time-to-market process. With a variety of capabilities that intricately ease the software development experience, Flutter enables developers to get their apps to market at record

speed, all while effortlessly distributing new features and upgrades to both iOS and Android platforms at the same time.

The efficiency provided by Flutter's uniform codebase across various platforms is critical in shortening the time-to-market process. Unlike the typical technique, which requires different codebases for Android and iOS, Flutter uses a single codebase to support both platforms. The harmonic cohabitation of code speeds up the development process by reducing the need for redundant coding efforts, allowing developers to focus on improving the app's functionality rather than grappling with platform-specific complexity.

12.2e Unified App UI and Business Logic Across All Platforms

Flutter has a compelling benefit in that it allows users to construct apps with a uniform UI and shared business logic across all platforms. Unlike traditional native apps that require different codebases for Android and iOS, Flutter developers may use a single codebase for both operating systems.

This not only streamlines development work but also guarantees that the user experience stays consistent across cross-platform mobile devices. Developers may decrease development costs, expedite time-to-market, and provide a unified app experience to their consumers by leveraging Flutter's features.

12.2f Hot Reloading

One of the most notable benefits of Flutter app development is its robust "hot reload" functionality, which allows developers to instantly view the changes they make in real-time. The feature allows developers to see real-time modifications in their program without having to restart it fully.

When compared to equivalent features in rival frameworks like as React-Native and Xamarin, Flutter's hot reload stands out owing to its incredible speed. The functionality dramatically speeds up the development process, allowing developers to quickly iterate, test, and fine-tune their code. As a consequence, Flutter developers benefit from increased productivity, shorter feedback loops, and the ability to fix issues quickly, making it a great choice for agile and dynamic development environments.

12.2g Shorter Testing Process

Flutter, a renowned mobile app development framework, has a special edge that speeds the testing process. The cross-platform nature of Flutter apps, which use a single codebase for both Android and iOS, results in a decreased testing effort.

Unlike traditional native app development, which requires separate testing for each platform, Flutter developers can complete full testing once. The effectiveness of this method not only reduces testing time but also the likelihood of encountering platform-specific difficulties. The user may assure faster releases, cheaper testing expenses, and a more uniform user experience across several devices by leveraging Flutter's accelerated testing approach.

The reason the team has also chosen to develop the AI Pothole Detection application using flutter is thanks to the multiple libraries and features that can be utilized with Flutter’s framework to develop this project:

TensorFlow Lite

TensorFlow Lite is a small form of Google's TensorFlow system made for embedded and mobile devices. It gives you tools to use machine learning

models on mobile systems, which means it works with Flutter apps. TensorFlow Lite models can be easily added to Flutter with the tflite_flutter package.

The ML Kit for Firebase

ML Kit is a Firebase SDK that makes it easy for developers to add Google's machine learning models to their apps. It has APIs for tagging images, reading text, finding faces, scanning barcodes, and more. There is a package called firebase_ml_vision that lets Flutter apps use the features of ML Kit.
Dart Packages: The computer language used by Flutter is called Dart. It comes with a set of machine learning and AI packages that can be used right in Flutter apps. Some packages, like dart-ml or dart-nlp, can help with different machine learning jobs, but they might not be as flexible as TensorFlow or ML Kit.

Hugging Face Transformers

Hugging Face offers cutting edge natural language processing tools and models that have already been trained. Even though these models are mostly reached through REST APIs, there may be community-made packages or tools that let you use them with Flutter apps.
You can get OpenCV for Flutter here: A well-known tool for computer vision and picture processing is OpenCV, which stands for "Open Source Computer Vision Library." Using the opencv_flutter package, developers can access OpenCV features in Flutter apps, which lets them analyze and change images.

Fastai

Fastai is a deep learning tool that makes it easier to train neural networks quickly and correctly. Even though Fastai is mostly used in Python, there may be active work to add its features to Flutter apps.

13 Mobile Application Architecture and Logic

13.1 Multi-platform vs Native Frameworks

The decision on which framework to use can ultimately make or break this entire project. It is a difficult choice because each framework and development kit will have their strengths and weaknesses that can either accelerate the development or stop it entirely. Not only that but there are un-written nuances and unexpected behaviors within those frameworks and kits that can only be discovered and solved in the deepest and darkest stack overflow thread from years prior to development.

There are many choices to make when developing a mobile application, first being should one use a development kit that is native to the mobile device’s operating system? Or should one use a multi-platform development framework? The pros of using a native language and kit is that the performance and compatibility of that application will be the best there is compared to a multi-platform application. Building an app in native code is guaranteed to run as it is intended on that operating system because it is known for a fact that the language is compatible. That is one of the downsides of using a multi-platform framework for development; since the user writes their code in one language and have it compiled down to a native language or run in a VM on that device’s operating system, there will be a sacrifice in performance and unintended behaviors that arise during the process of compiling the code to that native language, for multiple native

languages. On the other hand there are many limits to an OS native codebase, one being it could lack the libraries and packages needed to perform specific tasks that the application relies on to function. Since this application heavily relies on the logic of a machine learning model, it is required that any framework or development kit the group uses has libraries to integrate that model. Even if native development kits do have these libraries there is another obstacle with native development kits and that is the user needs to make a separate code base for each operating system. Two of the leading mobile phones on the consumer market consist of Apple Devices and Android Devices which have two completely different Operating Systems that use different languages.

Now getting on with multi-platform frameworks. That was briefly discussed the downsides of multi-platform frameworks in that there are unexpected behaviors and performance issues within them since they have to compile down to many different languages all from one code base. Even with that being said, there are still many more advantages with using a multi-platform framework. The biggest advantage is the efficiency in development. Having to write one code base that works on all platforms is probably one of the best tech innovations since the Motorola Razr flip phone. This advantage will reduce development time by fifty percent and that will mean there is more time to fine tune and test the model integration so it functions as intended. This advantage alone is enough to unanimously choose the multi-platform framework route. If the group was an organization consisting of multiple mobile development teams and had more than a dozen employees, it would be logical to create multiple code bases for mobile development. But the group is a team of 6 developers and the mobile team consists of 3 developers. It is not logical to attempt to create multiple code bases that perform the same functions. Besides that, the two leading

multi-platform mobile frameworks in the industry both provide packages and libraries to integrate trained AI models into a mobile platform.

13.2 Logic and Architecture

Even though web and mobile development may seem like a simple branch of the software development industry, mobile applications and their architectures can circumstantially be very complex. When not organized properly, the long-term development of a web or mobile application will eventually fail due to many holes in the structural integrity of the codebase. The application will not quite literally fail, but there will be unavoidable roadblocks in the future of development and an application will fail to be scalable for the business or organization that designed it. When designing architecture, there are 3 main principles to follow which are: SOLID, KISS, and DRY (marwaMejri, 2023).

SOLID is the first principle and it is an acronym for the five fundamental object-oriented design principles that were originally created by Robert C. Martin, also known as Uncle Bob (Mehta, 2021). “S” stands for “Single Responsibility Principle” and it states that a class should have no reason to change. What that means is classes should be very small and handle only one single responsibility (Mehta, 2021). In a general sense, a class should only take up about one digital screen of code. This allows classes to exist as an abstract tool used in the codebase; if a class needs to be edited, chances are there are too many different responsibilities given to it and it should be split into smaller classes. The “O” stands for “Open-closed Principle” which relates to single responsibility in that objects should be available for extension but will deny any modification (Mehta, 2021). When working in object oriented programming, it is important to utilize inheritance

and this rule focuses on the proper way to use inheritance. If a subclass extends a parent class, it should add attributes and methods that properly align and seamlessly build on top of that parent class, without any need for modifying that parent class on a codebase level or during runtime. Open-closed highlights the importance of using private variables and methods when designing each class. “L” stands for “Liskov Substitution Principle” which is the principle of keeping consistency throughout inheritance. Any parent class that has extending subclasses, should be easily replaceable by a relevant subclass without causing any system errors. This entails designing subclasses to function and behave in the same exact way as the parent class. “I” represents the principle of “Interface Segregation” which also relates to Single Responsibility where interfaces should be small and concise. Classes that implement interfaces should not need to implement a large interface that has many methods not being used by that class. Instead of having one main interface with loads of service methods, developers should create multiple interfaces with more refined methods for a specific purpose so that they will individually be used by different types of classes. This is a great construction of proper abstraction; if one interface needs to be modified, it will only affect a small number of classes. “D” stands for “Dependency Inversion Principle” which says higher level modules should not depend on lower level modules, each of them should utilize abstraction (Mehta, 2021). This principle relates to the other principles in emphasizing abstraction in order to build a modular system. Large scale classes and small scale classes should be decentralized and not have a dependency on each other whereas if one were to change, it will not directly affect the other.

KISS is another software design principle which, in some ways, contradicts SOLID. KISS stands for “Keep it Simple, Stupid” and as one may assume, its primary focus is to use processes that ensure developers are not

overcomplicating the codebase. Which does contradict SOLID in ways because SOLID has many principles that require keeping class files small and creating a lot more of them so they have specific responsibilities. Well a codebase with over two hundred class files is not one that is labeled as simple. It does however agree with SOLID because the purpose of it is to keep classes and other codebase files very small so they are easily readable and serve one specific purpose. KISS also calls for removing redundant methods in the codebase or relocating them where they are useful and also eliminating code clutter (baeldung, 2023).

The third major principle of architecture is called DRY. DRY stands for "Don't Repeat Yourself" and the overall purpose of this principle is to reduce as much repeated logic as possible from the codebase. This does not refer to repeated code on a syntax level, but is more of a critical thinking and algorithmic perspective on classes and methods. A perfect beginner's example of this is something all developers learn the first time they are introduced to functions in programming. The purpose of a function is to write a specific process of code that has a predicted known input and an expected known output based on a certain logic. The purpose of writing a function is if there are multiple areas of the codebase where certain commands are repeated and identical to one another, so a function is created and called in those areas to reduce code clutter.

While there are many different methods of designing architecture for a mobile application. This section will focus on the proper structure and organization of a Flutter application, more specifically, a Flutter mobile application. The method to design architecture for the AI Pothole Detection mobile app is called clean architecture. Clean architecture focuses on a principle called separation of concerns (Flutter Guys, 2023). This involves dividing the software into layers that are designed to be modular. Which allows for a simple development process since each layer can be developed

individually and updated or reused efficiently. The overall goal of clean architecture is to be scalable, readable, testable, and easily maintainable. From a top-down perspective, a Flutter application will have 3 main layers. These layers are the Data, Domain, and Presentation layer. The Data layer will not be discussed in this section of the design document because it is unnecessary for the primary function of the team's bare bones application. However it is a requirement for the stretch goals of the AI Pothole Detection project.

13.3 Presentation layer

The presentation layer is the simplest layer of the application because it contains all of the visual aspects. The presentation layer is where all of the app's User Interface elements reside such as pages and widgets that exist within those pages. The presentation layer does however contain a lot of logic with state management. State management is the process of managing the states of each page and widgets within those pages so the user interface items can be updated upon user interaction. State management is also responsible for triggering requests to other application layers such as the domain layer or the data layer, when a specific state is achieved for a certain widget. The second responsibility is more of a circumstantial event and not every widget will be required to make requests depending on its state. State management is a big deal in application development for every kind of mobile application, not just Flutter. There are many ways to perform state management in a Flutter application, the most efficient way is to use a state management library. With that in mind, there are many different state management libraries to choose from and they all handle state management differently. The entire goal of state management is to use abstraction to remove the current states from the widgets and relocate them in their own node so that they can be accessed by other widgets and other layers of the

application. For ease of use, the team will use Flutter's riverpod widget state management package. Riverpod works on a publisher and consumer system. Riverpod has nodes called providers which will hold the state of a specific widget or grouping of widgets and will continuously be updated by those widgets. Then from there a developer can create a consumer to request the state of a widget from that provider. This way if widgets from other classes or other pages rely on the state of a certain widget, they will have global access to that state.

The entire app on a foundational level will have three pages in the presentation layer: A home page, camera view page, and a settings page. The home page will be a pretty basic page with static information and one button which will allow the user to navigate away from the page. The home page will not have any state management within it due to its simplicity. The camera view page is the bulk logic of the application. It will contain a few widgets. The first widget is the camera view, this widget will fill the entire screen, excluding the navigation bar, and will provide a live camera stream of the smartphone's rear camera. This camera stream is important for the viewer to understand how the app turns the phone into a dash cam and it will also help visualize where potholes are spotted, using these next widgets. The other two widgets will be banners that pop up on either side of the screen when a pothole is detected. If a pothole is detected on the right side of the camera view, it will pop up on the right side of the page, and vice versa for the left scenario. There will be a lot of state management within this page. The camera view widget will have a provider for its states which will also connect to the model integration class that exists within the domain layer (that will be discussed later in this document). The two banner widgets will consume from the camera view's provider and their states will change based on the state of that provider. The third page will be a settings page. This page will have a collection of widgets that will be a collection of toggle

switches, navigation buttons, and slider bars that will control the behavior of the app's features. It is uncertain on how many widgets it will have at the end of development but here is a quick list of the buttons deemed necessary in the design phase. There will be toggle switches for on screen pothole notifications and sound alert notifications, a slider bar for sound alert volume, and a button to navigate back to the home screen which will double as a more information screen. State management for this state will be a settings provider that will wrap all of the widgets on the settings pages mentioned above. Every widget on the app that relates to any setting in the settings page will consume the states that exist in this settings provider, so that their behaviors reflect those states accordingly. For visual representation of this layer, There is a diagram included in this document for reference, in the Figure 13.3.1 below.

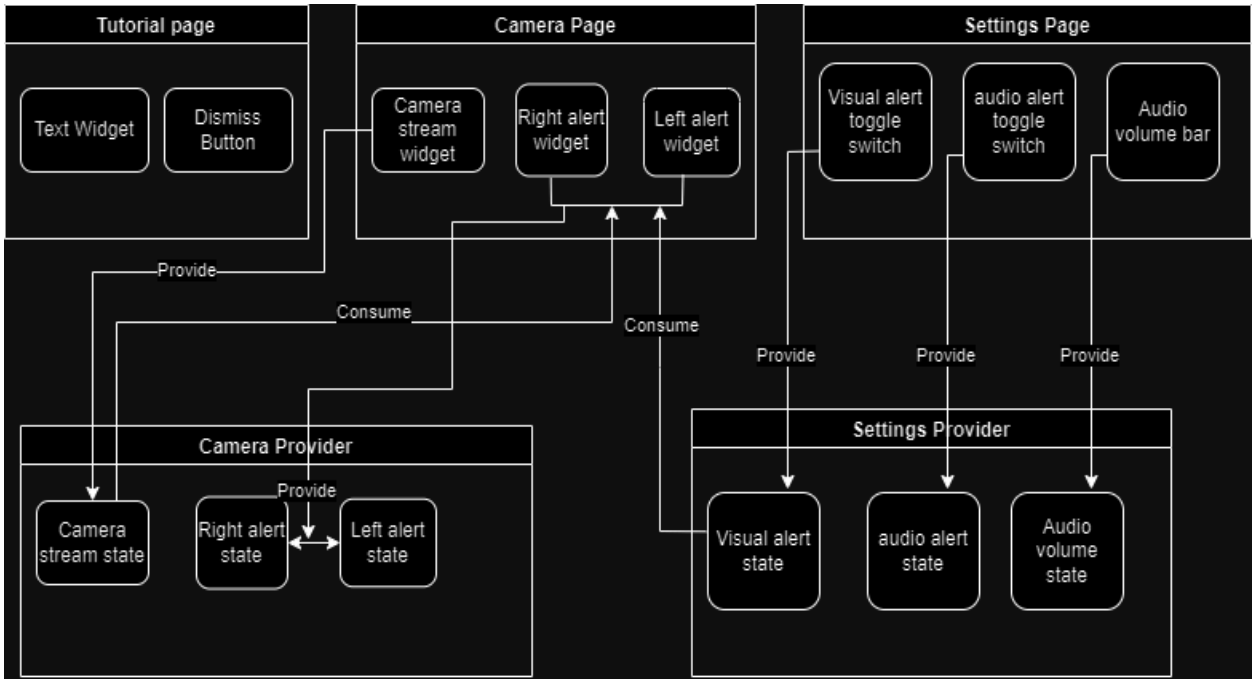


Figure 13.3.1: Presentation Layer Diagram of Mobile Application

13.4 Domain Layer

The domain layer is an abstract layer that has no dependency on other layers like the presentation or the data layer. This layer contains all of the heavy logic of the application and one might say this is the bread and butter of the application. The domain layer will consist of class files that handle different aspects of logic of the application. The purpose of the domain layer reiterates the standards of clean architecture in that it should be modular, scalable, readable, testable, and easily maintainable (Flutter Guys, 2023). The domain layer will be written entirely in dart and not contain any Flutter library code. The domain layer consists of three main parts: entities, repositories, and use cases.

Entities are essentially the business objects of the application and the most high-level rules end up being encapsulated within each entity (Max on Flutter, 2023). To give an example of how one would picture this, entities would be very similar models that are class objects in an Object Oriented web application like one built in Java’s SpringBoot. Entities should only exist to hold values of certain models and interact as objects, they should often not perform any implementation such as logic for said entities, that is the responsibility of use cases. The only case when an entity can contain logic within it is if that logic handles how that entity object can be modified or how it behaves when being used in a use case.

Repositories are an abstract collection of entities within the application. The repository is similar to a service layer or a controller layer in a web application, where it aligns with entities and allows CRUD operations to be performed on the entities that are stored in the data layer (Max on Flutter, 2023). A repository is an interface for an entity and they have a single responsibility which is to provide and manipulate data for the entity that implements it during runtime.

Use cases handle actions performed within the application during runtime (Max on Flutter, 2023). Use cases are classes that encapsulate rules and procedures of entities so that they can be manipulated and evaluated within other parts of the application. They are similar to repositories in that they are specific to certain entities and their repositories but they handle the logic of specific scenarios that the entity will encounter with user interaction during runtime.

With that being stated, the AI Pothole Detection system's back end logic is fairly short and sweet. While it does perform an intense amount of precise calculations, those are all handled by the Yolo model that will be built by the machine learning team. With that black-boxed, the mobile team will create one entity with a repository to go with it, as well as many use case classes to handle actions. The entity for the neural network will hold all the values of the model, whether that be stored in the codebase itself or in a database remotely. Then a repository will be created which will handle the logic of loading and running that model. Live camera stream footage can then be inserted into that running model during runtime. The team will then create several use cases that will be called by the repository and each use case will have a specific action to return and manipulate the state of the widgets in the presentation layer based on the values returned by the model. To wrap up architecture, an activity diagram (Figure 13.4.1) and a block diagram (Figure 13.4.2) are provided below to give a visual representation of how the logic will work with entities, and use cases of each feature in the codebase. As well as a higher-level overview of each system and how they interact. Keep in mind these diagrams are designed for the stretch goals of the project and will contain some nodes not discussed in this section.

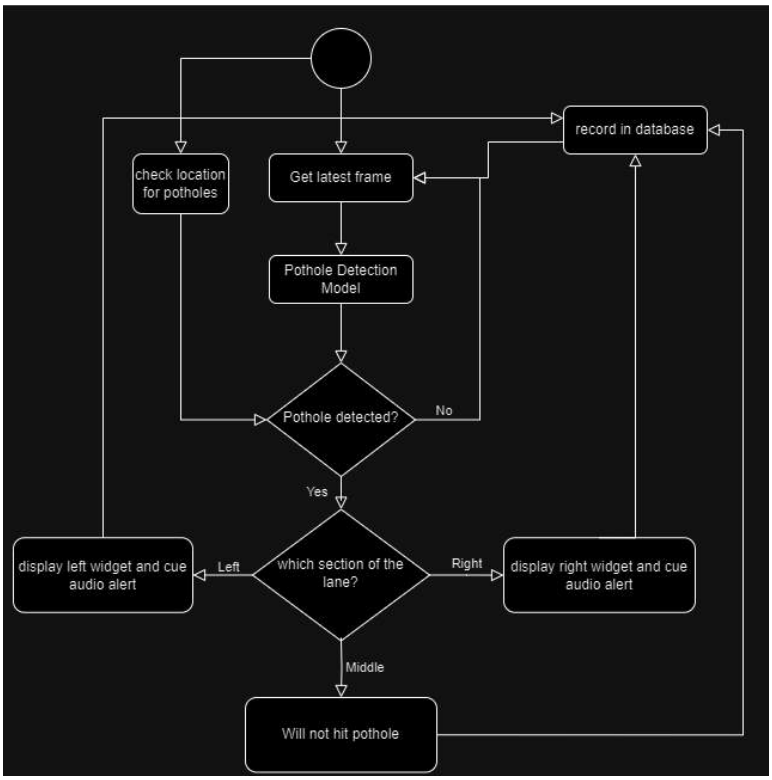


Figure 13.4.1: Activity Diagram for Mobile App

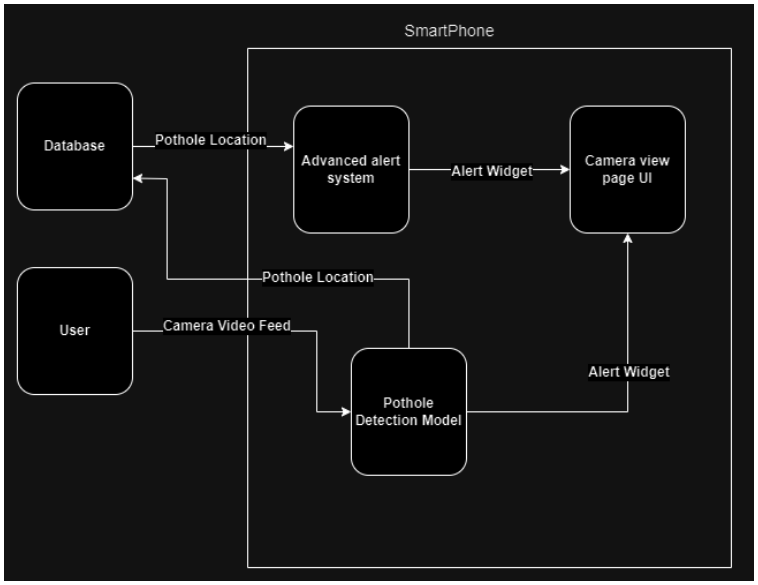


Figure 13.4.2: Block Diagram for Mobile App

13.5 Model Integration

Integrating the Yolo model into the flutter application is the most important part of this project. Without model integration, this project serves no purpose whatsoever, this integration is the mobile team's biggest hurdle. Unfortunately there are only a few common methods of implementing a pytorch machine learning model in a flutter application which limits the options of the mobile team in some ways. The best method discovered so far is to utilize the flutter_pytorch package to run the model.

First, the team will install the flutter_pytorch package and list it on the yaml file containing all of the codebase dependencies. Then once the foundation of the app is complete, and the model is trained, the domain layer of the camera page feature will contain a model entity. Then specific variables need to be declared in order to load the model. Once the variables exist, a load method will be created in the camera page repository. Once the load method has been created, it will be called in the init_state() method of my_app() in the main class. This will ensure the model loads on startup. Once that is done, all that is left to be created is a pothole_detection() method that will take the camera_feed as a parameter and run the model with the frames of the camera feed. Then the logic of the function will call use_case methods to edit the states of the alert widgets depending on the output of the model. Figure 13.5.1 below describes the use case diagram for the user and database interacting with the mobile app.

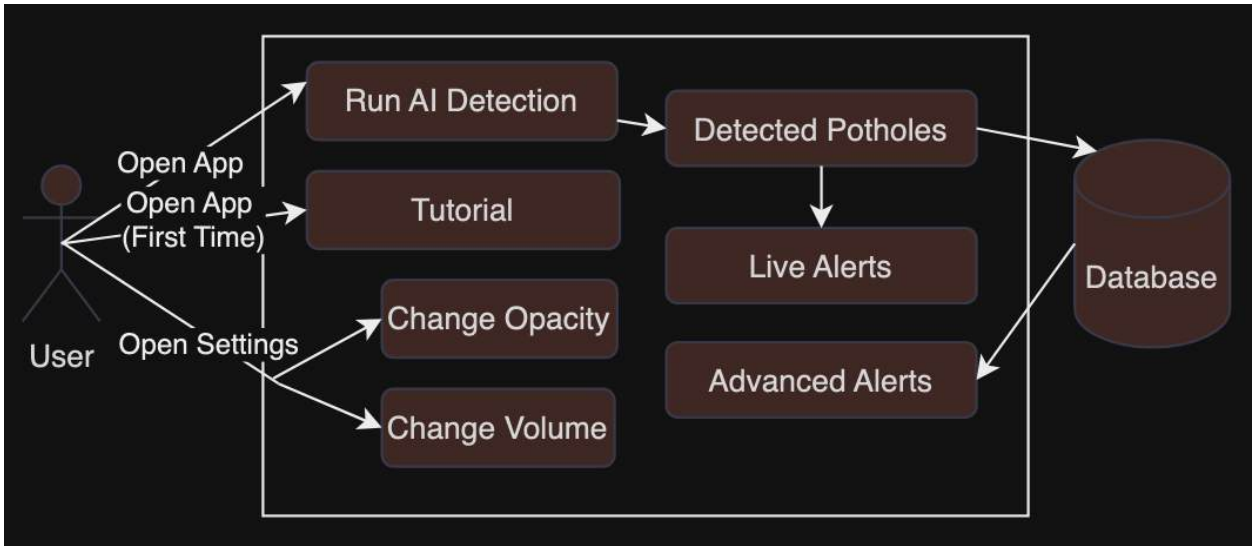


Figure 13.5.1: Use Case Diagram for Model/Mobile Integration

The team ended up implementing multiple different AI models into the mobile application. Sadly, only one model was able to reach the minimum required threshold of 5 frames per second. Mobile Net was this model and with the fact that this is the smallest model we could afford as well as the implementation of quantization the detection algorithm efficiency of the model dropped drastically. After trying multiple models and being unsuccessful with being able to get performance that would be considered acceptable the team ended up moving to a desktop application that would utilize a laptop in order to run a larger model in order to get the performance we required.

13.6 File structure

Flutter file structure is quite different from many other application development frameworks and are triggered from the repositories circumstantially on a case-by-case basis. That will wrap up all of the files in the codebase and how they are structured.

14 Laptop Program

14.1 Motivation

The laptop program was developed after the mobile application specifically because the mobile application was not able to run an acceptable model in real-time harming the model performance and causing harm to the device itself through overheating and throttling. Therefore, the team decided to develop a laptop-based program in order to have a similar experience as the mobile application, but be able to access the hardware power available on a laptop. The team also decided that, because laptops are not easily deployable by normal drivers, the project would be more oriented towards municipality usage. Therefore, the laptop program would still contain the same core functionality as the mobile application did (real-time detection, directional alert system, lane detection, etc.), but the mapping system would be more robust and have more features than the initial mobile application was planned to include.

14.2 Hardware Used

The primary testing hardware the team used for this application was a Dell G5 Gaming Laptop with an RTX 2060 GPU. This device was used as Nicholas Gray, the head of the AI team and the person developing the laptop program, already owned the device and knew it would be able to run the YOLOv8 model in real-time. However, as the device does not contain a GPS antenna, to gather the GPS coordinates of the potholes during live testing, a Samsung Galaxy S10+ was also used concurrently, also owned by Nicholas Gray.

14.3 Language and Libraries

The laptop program was implemented entirely using Python because of the requirement to use an AI model and the speed of development and deployment compared to other languages, as discussed previously in the AI section. The main GUI was developed using CustomTkinter, which is an extension of the original Tkinter GUI library developed for Python. CustomTkinter's main advantage is that it provides a more modern looking UI while preserving the original functionality of the Tkinter library. The motivation for using the Tkinter library over other Python GUI libraries, such as PyQt, was due to the need to make the laptop program in less than a week. Therefore, the team decided to use the library they were most familiar with that they knew would work for the program. The advantage of using Tkinter is that it is very easy to initially set up and use, given its simple API structure. However, the main disadvantage is that it is quite antiquated and difficult to use to develop more complex applications, which the team encountered while developing the laptop program.

For the deployment of the AI model, the team used Ultralytic's YOLO library, which allows for very simple loading and inference of YOLO based models. For streaming images into the model and manipulating them for detected potholes and later alerts, the team used OpenCV. OpenCV was used because it is the most popular library for image processing and computer vision tasks in Python, and integrates well with the YOLO library.

For displaying the map in the program, the team used TkinterMapView, which is a library designed to allow users to add a simple Tkinter map component to their code (Schimansky, 2024). TkinterMapView allowed for direct integration of OpenStreetMaps into the application, and allowed for potholes to be added to the map[as markers, left and right click map

functionality, bounding box selection functionality, and tying metadata to the markers on the map.

For geolocation, a smartphone has to be used with the IP Webcam app (Khlebovich, n.d.). This app allows someone to convert the smartphone into a network camera that can be accessed from a HTTP IP address. This can be potentially used as a camera source for the laptop program by streaming from the camera to the laptop. The app also can be enabled to share the GPS location of the smartphone, which is accessed from an API endpoint on the same IP address as the camera stream. This was the primary use of the app for the laptop program, as there were no other convenient means to gather the GPS coordinates. The team explored using a USB GPS antenna, but the device used did not work and was returned.

For finding the address of the potholes, the GeoPy library was used, with the Nominatim system used to take the location and find the closest address. This system was used as it is a free API endpoint as it is relatively simple to use directly through the GeoPy library. The downside of using this system is that it is less accurate in its reverse geocoding compared to a Google Maps service, and provides an inconsistent return of the address. For example, some locations will return a direct address, but others will just return the street name or just the county.

14.4 Overall System Design

Figure 14.4.1 shows the overall activity diagram for the laptop application. The user gives a video feed directly to the AI model. As the AI model operates, it looks to see if it detects any potholes. When it does, it feeds that to the side detection system. If the pothole is within the lane, it detects whether it is on the left or right side. It sends this directional

information to the alert system, which then displays an alert to the user. At the same time, it pushes the pothole coordinates to the database via an API. To get the GPS coordinates of the pothole, the program fetches the GPS coordinates from the connected smartphone using the IP Webcam app. The side detection system will also push up an image of the detected pothole so that users can later see what the detected pothole looks like. When using the mapping program, the map fetches the pothole locations from the database. This map displays the potholes as markers, and links the images directly to the potholes. The user can also add and delete potholes directly from the database using the map. Finally, the user can download the potholes and their locations and addresses from the map itself, if they desire.

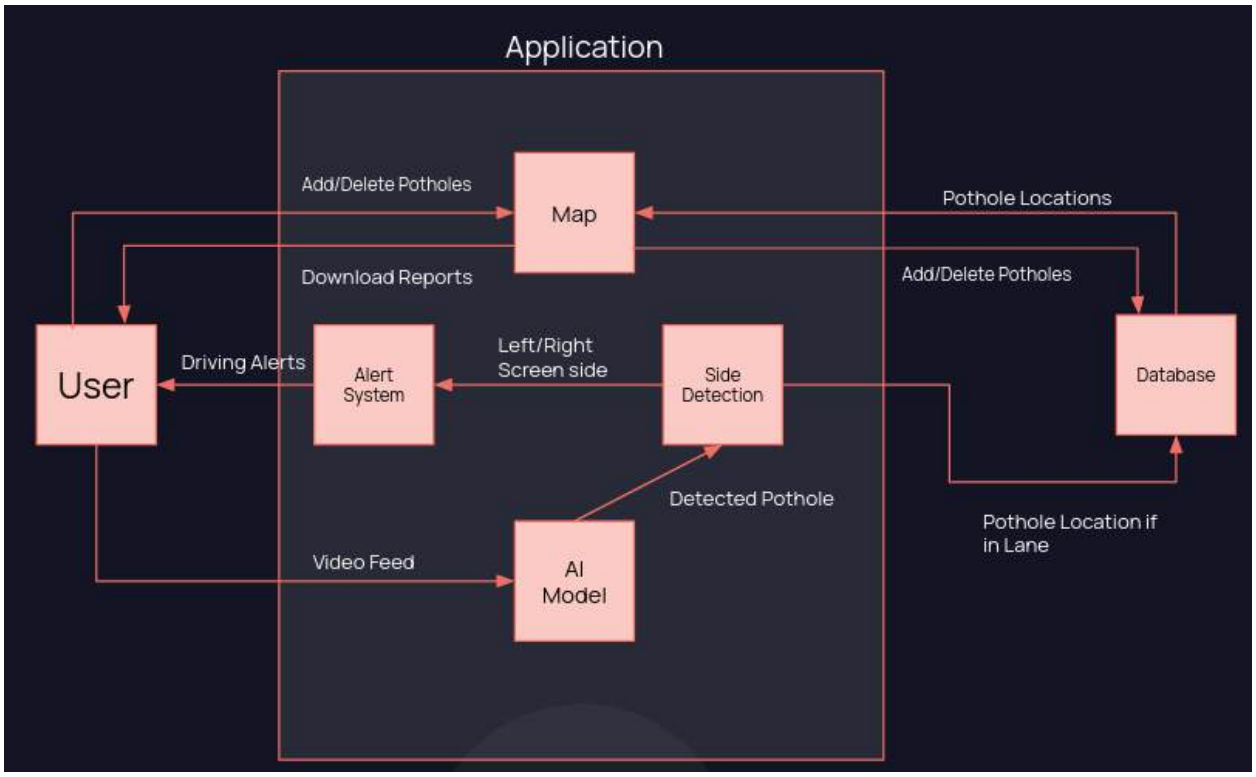


Figure 14.4.1: Activity Diagram for Laptop Application

14.5 Initial Startup Screen

When the program is first started, an initial start screen is shown with options to select the video feed and the used AI model, as seen in Figure 14.5.1.

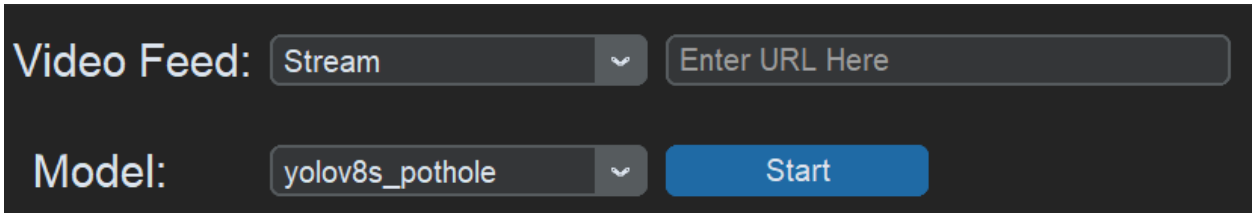


Figure 14.5.1: The initial laptop start screen.

For the video feed, the options are a video stream and the available cameras connected to the device. For the stream, the URL is expected to point to a continuous MJPEG stream, which OpenCV can read directly. The disadvantage of using a video stream is that the stream will be at a constant 30 FPS, while the AI model will lower that, leading to a buffer in the video stream. This buffer means the program will lag behind the live stream, making it not real time. To counteract this, the live stream has the queue limited to 1 frame, but this caused bugs elsewhere in the program. For the cameras, the system finds them by cycling through each camera by its index and seeing if OpenCV can open them. This is a relatively direct approach, but the disadvantage of this is that if there is a camera opened by another program, OpenCV won't be able to open it and so it and all cameras with a later index will not be listed. This is particularly problematic if the laptop integrated webcam is being used, as that often has a camera index of 0. Therefore, this program should be used with no other program possibly using the camera.

For the model, the program looks for any ".pt" file that is in its local directory. The models it finds have their direct paths saved internally in the code, which allows for other models to be loaded later in the program if the

user wishes to switch which model they are using. Currently, the only model available is “yolov8s_pothole”, which is the YOLOv8 model the team trained that performed over the YOLOv5 and MobileNet models.

14.6 Video Feed Page

After pressing start in the initial window, the main program will load. The main program has two tabs, “Video Feed” and “Map”. “Video Feed” is where the pothole detection occurs and is the main page of the program. In the Video Feed page, there are three options: Pothole Detection, Lane Setting, and a Settings button. The Pothole Detection toggle turns the AI on or off, which allows the user to set up the program without having the possibility of sending any false positive potholes up to the database during initial setup. The Lane Setting switch turns on the visibility of the lane detection system, which is running at all times but has a default configuration that can be adjusted by the user. The purpose of the lane detection system is to focus the area of possible potholes being detected to an area specified by the user, which is expected to be the road the user is on or the lane itself. The following section will go into the lane detection system and its interaction with pothole detection in greater detail. The final option on the Video Feed page is the settings button, which opens a separate settings window that allows for adjusting settings for the AI model, alert system, and video feed. Figure 14.5.1 shows the entire Video Feed page and a demonstration of the lane detection system being set up.

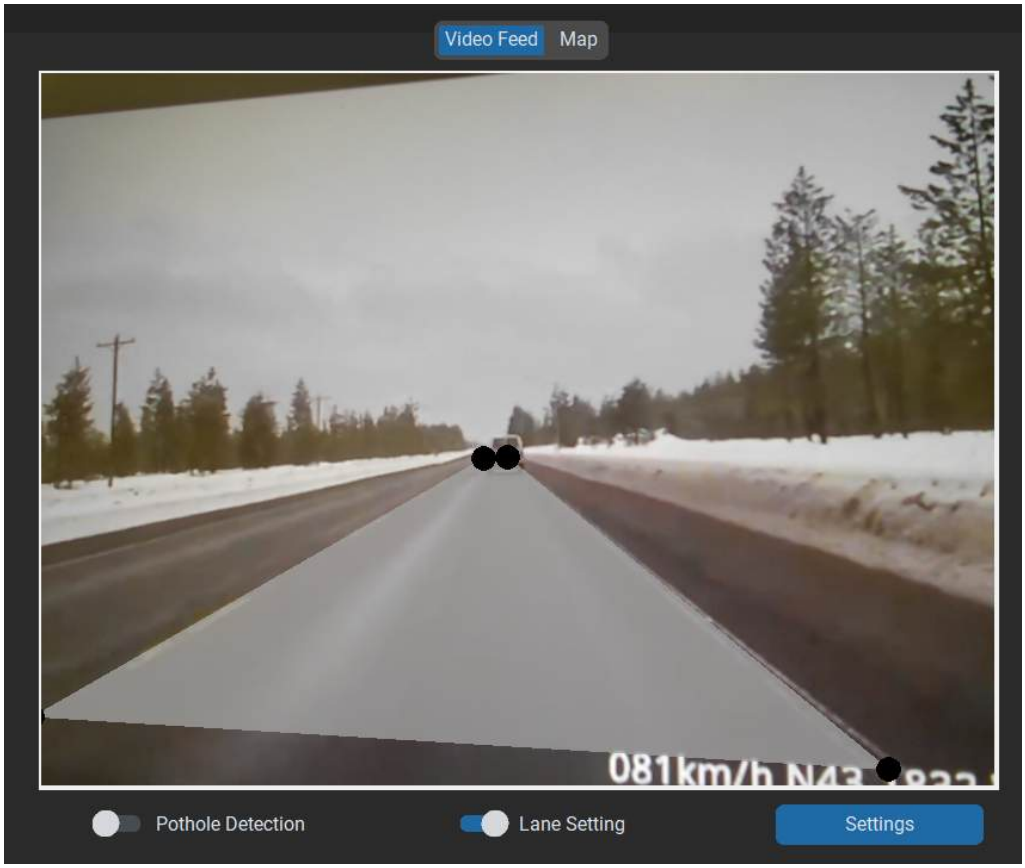


Figure 14.6.1: View of the Video Feed Page

14.7 Lane Setting System

As can be seen in Figure 14.6.1 and mentioned previously, there are controls for setting up a lane detection system. The lane detection is controlled through four points directly displayed on screen that form a quadrilateral around where the user is interested. Internally, this quadrilateral is vertically split into two further quadrilaterals to detect the "left" side and the "right" side of the detection plane. This "left" and "right" side was created in order to fulfill the need for having directional alerts to tell the user which side of the lane the pothole is on. This does mean that the alert system is primarily designed with the idea that the lane setting system will be set up on just the lane itself, but the system does not prevent a wider view from being set.

As potholes are being detected from the live video feed, the center of the bounding box of each pothole is found. If the center is within one of the quadrilaterals mentioned previously, then an alert is sent and the pothole is uploaded. The point is detected if it is inside the quadrilateral by loading the four points of the quadrilateral as a polygon, and then doing a point test to see if the point is inside. This point test is done by drawing a ray from the point out of the polygon. If the point hits the polygon an odd number of times, it is inside.

14.8 Alert System

For the alert system, it is similar to how it was done for the mobile application. After the program detects a pothole on either the left or right side of the lane, it internally sets a flag to display the left or right visual alert for a set number of seconds. The system may also play an audio alert, if the user has it turned on. During that time, a red box is displayed on either the left or right of the screen, similar to how it is done in the mobile application. At the same time, the program sets a different flag to ignore any further alerts from the program for that side of the lane for that many seconds. This does not prevent the model itself from doing the detections, nor does it prevent the bounding boxes from being displayed, but it does prevent the alerts from calling the API endpoint for pushing to the database. Like in the mobile application, the purpose of the delay is to make sure that the program is not pinging the API endpoint multiple times for a single pothole that it may see across multiple frames. The delay system is also set up to make sure it doesn’t push for a cluster of potholes. After the alert expires, the visual alert is no longer shown and the system allows for alerts to be sent from that side of the lane again.

14.9 Pothole and Image Uploading

After a pothole is detected and the lane detection system confirms that it is within either the left or right side of the lane, the uploading system starts. First, the program calls the API endpoint from the IP Webcam app in order to fetch the phone's GPS coordinates, which should be near the laptop or the phone stream being used to view the road. The API endpoint looks something similar to "http://<IP Address>:8080/gps.json". This endpoint returns a JSON response in the format of

{"gps": {"latitude" : X, "longitude": X, "altitude": X, "accuracy": X}, "network": {"latitude" : X, "longitude": X, "altitude": X, "accuracy": X}

, where "gps" is the GPS coordinates from the GPS antenna on the phone, and "network" is the GPS coordinates of the network device the phone is connected to. In the code, the program first checks to see if it can access the longitude and latitude from the "gps" return. If it exists, the longitude and latitude are saved. Otherwise, it similarly checks for the longitude and latitude in "network". If neither can be accessed, then the program uses a default set of coordinates as the pothole location.

After the GPS coordinates are returned, the program POSTs to the pothole upload API endpoint. This endpoint will either create a new pothole at that location, or increment the count of an existing pothole in the database. The way this is determined is if the pothole is near another pothole that already exists in the database. If the pothole being uploaded is within 150 meters of an already existing pothole, then that pothole's counter is increased; otherwise, a new pothole resource is created in the database. The API endpoint then returns an ID that represents a pothole in the

database now corresponding to the GPS coordinates POSTed if the upload is successful.

With this pothole ID, the program then calls on the image upload API endpoint. When the upload system is called, the program passes an image of the current frame of the video stream to the system, which is expected to contain the actual pothole in the image. This image is then resized to a quarter of the original, then encoded into the PNG format, and then finally encoded using Base64 encoding. The first encoding step is done so that the image can be stored on the Supabase database directly as an image. The second step is done in order to make it easier to upload the data to the API endpoint. The thought process behind this was that uploading the image data directly makes the endpoint more complicated and there might be slowdowns with uploading the image. On the other hand, using a base64 encoding means the image can be represented as a string and be uploaded in a JSON, which makes the API endpoint more understandable. The image upload API endpoint uses the encoded image and pothole ID as JSON inputs, and returns a response code for if the upload was successful or not.

The entire uploading system is called on a separate thread from the main program. This is done because calling the APIs is a blocking call on the main thread, so if they were not called in a separate thread, the entire program would freeze while the upload system completes. This would be a major problem if any of the API endpoints timed out, as this would cause the program to freeze for 30 seconds or more. This system also allows for multiple API calls to be made in the background without any concern for them blocking each other.

14.10 Settings

Like in the mobile application, the laptop program has several settings for the model, alert system, and the video stream. These settings appear on the Video Feed menu when the Settings button is pressed, which spawns a separate window. This window does not stop the main program, which allows for the settings to be adjusted in real time while the program operates on the road.

14.10.1 Detection Settings

On the detections tab of the settings page, there are three main settings: confidence, show detections, and model. Figure 14.10.1 demonstrates what the Detection tab of the Settings window looks like.

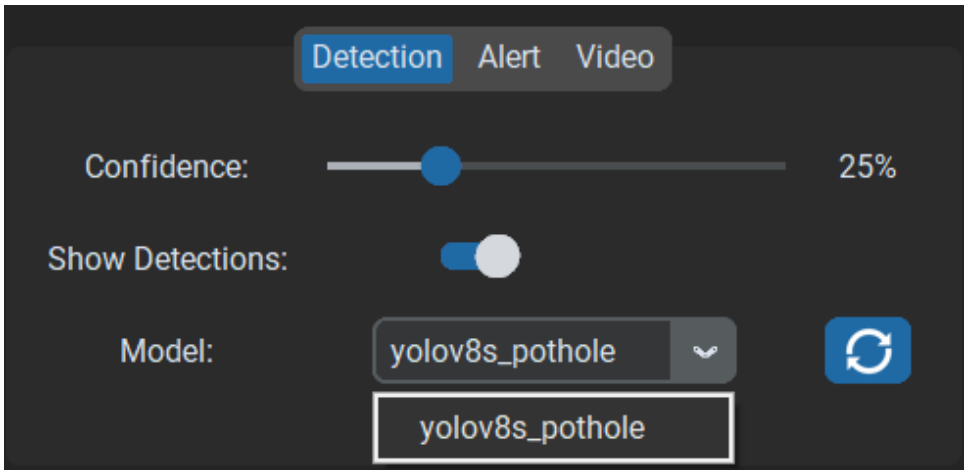


Figure 14.10.1: Detection Tab of Settings window

The confidence setting controls the minimum confidence of the model in order to return a pothole. In object detection, a bounding box not only returns the object location and class, it also returns a confidence score on how confident it is on the detection. This confidence score can be used to filter out any predictions that the model is not confident in, which are often false positive predictions of potholes. The default confidence is 25%, which

through testing was found to detect most potholes and not detect too many false positives. However, on certain roads where false positives are prevalent, the confidence can be increased, decreasing the number of potential false positives while at the same time increasing the chances of a real pothole being missed.

The Show Detections toggle simply controls whether or not the bounding boxes are displayed on the video feed. The bounding boxes are still detected and used internally for the alerts and upload systems, but it may be possible that the user does not want to see the detections themselves.

The final option, “Model”, allows the user to change which AI model they are using from a dropdown. As described previously, the program first scans through all the models that can be seen in the local directory of the program, specifically looking for the “.pt” files. The refresh button to the right will perform the same action and look for newly added models. When selecting a new option in the dropdown, the program will then unload the previously used model and load the new model, unless the new model selected is the same as the current model.

14.10.2 Alert Settings

The alert settings are similar to the ones found in the mobile app, but with some added functionality. First, the alert sound system is a single on/off toggle, instead of a volume slider. This was done because Python does not have a convenient means to control the volume of an audio file, and instead just plays it at the volume it's at. The next setting, alert opacity, controls how opaque the alerts appear on the Video Feed page, similar to the mobile application. Alert delay controls how long the alerts display for, and control how long the program should wait before it can push any detected potholes

to the database. Finally, Alert Width controls how wide the alerts are horizontally on the Video Feed page. This setting was added because the user of the program may not want to obstruct the view of the video, or they may want to have clear visibility of the alerts. Figure 14.10.2 shows the alert tab in the Settings window.

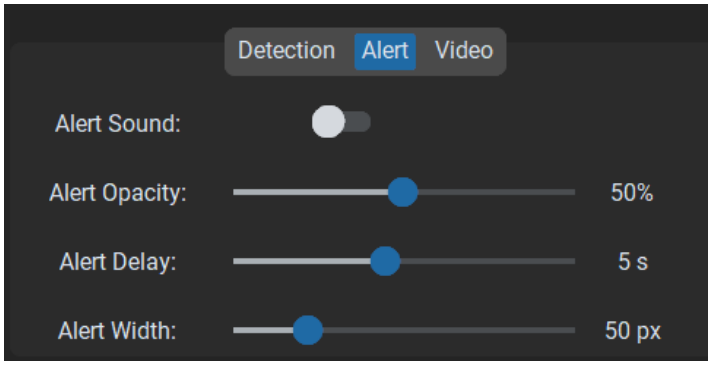


Figure 14.10.2: Alert Tab of the Settings window

14.10.3 Video Settings

The video settings is the final tab in the settings window that includes miscellaneous controls for the video feed. The first setting controls whether the FPS is displayed in the top left of the video feed for debugging purposes. The Show Lanes setting controls whether the Lane Detection system ever displays itself, regardless of the user turns it on or off. Finally, the Cameras dropdown, like the model dropdown, controls which video feed is being used. The refresh button works like the initial start of the program where it cycles through the available cameras in ID order, and stops once it can’t load one. Selecting a new option in the dropdown changes the camera. For the stream selection, if it is selected, a pop-up window appears, asking the user to type in a URL for the program to use as the camera feed. Figure 14.10.3 shows the Video tab of the Settings window.

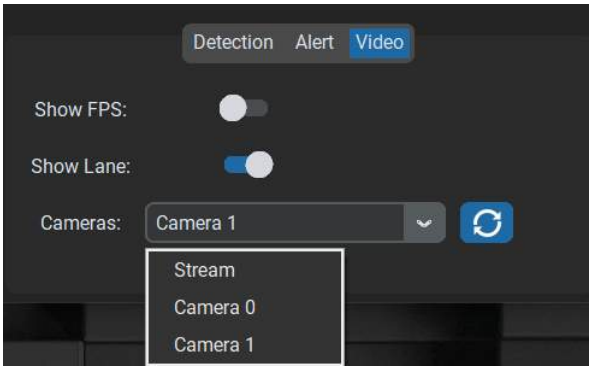


Figure 14.10.3: Video Tab of Settings window

14.11 Mapping Page

The map page is the second main component of the laptop program. The map was primarily designed around the use case of governments and municipalities, as it directly allows the user to control the potholes that appear on the map and add / remove potholes directly. Figure 14.11.1 shows an overview of the entire program, which the individual components will be talked about in the following subsections.

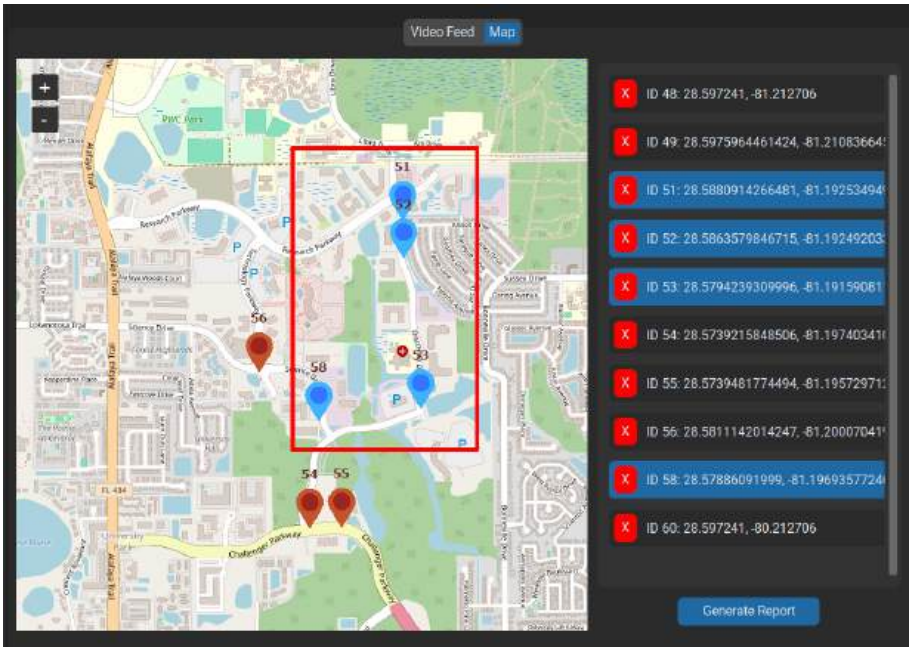


Figure 14.11.1: Overview of Map page

Like previously mentioned, the map is displayed using TkinterMapView, and when the potholes are loaded onto the map, each pothole also gets a tab in the side bar that allows the user to easily delete potholes. There is also a refresh button for the map, which clears all the existing potholes from the program cache and calls the API again to get fresh potholes. This is useful for someone else if running the program and adds new potholes, or when after doing collection the user wishes to view the newly added potholes.

14.8.1 Pothole Images

A useful feature of the map is that the pothole markers also have the images from the database associated with them. When a user clicks on one of the markers, the image is displayed over it. This allows the user to check if the pothole was properly detected by the program, or if the detected pothole is really a false positive. Viewing the image also allows the user to view how bad the pothole is, allowing municipalities to make judgements on which potholes are the most important to address.

14.8.2 Bounding Box Marker Selection

Another useful feature of the map program is the ability to select potholes using a bounding box. The user can draw a bounding box over the map by holding control and dragging their mouse over the map. After the user lets go of the left mouse button, a box is generated and the potholes inside the box are highlighted. The highlighted potholes' markers and their associated side labels are highlighted blue. Once the user selects the potholes, if they choose to generate a report, then only the selected potholes are used in the report. To clear the selection, the user just has to move the map and the bounding box selection disappears.

14.8.3 Adding / Deleting Potholes

As another feature for municipality use, the user of the laptop program can add and remove potholes directly from the map page. To add potholes, the user can right click on the map where they would like to add the pothole, which spawns a small right-click menu in which “Add Pothole” can be selected. This sends an API call similar to the one in the Video Feed tab to upload a new pothole. If the upload is successful and generates a new pothole, then the marker is added to the map. If the pothole is merged with another existing pothole or fails to upload, the marker is not drawn on the map. The purpose of this feature is to allow users / municipalities to add potholes they receive reports of but have not detected in their driving. This is added to minimize the inconvenience of having to drive out and detect every single pothole, especially if a report of one they missed is received. It should be noted that this feature does not provide any images of the added pothole, and so no image will be displayed when the marker is clicked.

For removing potholes, they can be removed simply by pressing the red X button on the side view for whichever pothole the user would like to delete. When this button is pressed, an API call is made to the database to remove the pothole with that ID. If the API call is successful, the marker is then removed from the app. The purpose of this feature is to primarily remove any false positive detections from the database and map. The false positives can easily be checked by clicking on the marker and viewing the image to see if there is actually a pothole in the image. Another use of the delete functionality is to remove potholes that have been patched, and therefore should no longer be in the database.

14.9 Report Generation

The final feature of the map page in the laptop program is the report generation. When the user presses the Generate Report button, the program

first goes through each pothole and performs reverse geocoding on it to find the closest address to the pothole location. If it cannot find an address, it simply returns the longitude and latitude as a string. The tool used for reverse geocoding is GeoPy with Nominatim. After the addresses are found, they are all added to a CSV file along with the pothole ID and its longitude and latitude. The user then has the option to set where the CSV file is saved and what the file name is.

This feature works directly with the bounding box selection feature. Like mentioned previously, if there is a bounding box selection on the map, then the report generated will be made with only the selected potholes. If there is no selection, then every pothole that is on the map - and therefore every pothole in the database - will be saved into the report. This was added to allow the user to generate a report on only potholes they specifically are concerned with, or to know where every pothole is in terms of its address.

15 Database and API

15.1 Introduction

Beyond creating an initial AI model capable of detecting potholes, we had a suite of extended features surrounding a database and API. These goals were referred to as “stretch” goals originally, and largely surrounded the implementation of the mentioned backend components.

This part of the project had two parts: the backend setup and the frontend enhancements. Many of the downstream frontend enhancements are mentioned elsewhere in the document, so we will largely be focusing on the backend setup and design choices under the hood.

The backend is largely composed of two parts: the database and the web API. The database the team decided to choose is a PostgreSQL database provided by Supabase. Supabase provides a fully kitted client library with all the needed tools for implementing an API that can populate and manage the database. Additionally, Supabase manages the hosting of the database, authentication, and serverless services, alleviating the need for micromanagement of deployments. Part of the group's ethos is leaning on established tools and companies to make tasks easier, and choosing Supabase is an excellent implementation of this concept.

Supabase uses a PostgreSQL database, a relational database with many useful features. As this project is managing some amount of user information, securing it is a primary concern. The database’s primary purpose is to store identified potholes as marked by the AI model. This task is not a simple one. An immediate problem is the possibility of error in the AI model where a pothole is marked “real” when it is not. To a degree, this event is inevitable with enough usage in the field. No AI is “100%” accurate and errors are possible even in the most sophisticated of AI models.

Potholes are a collection of many unconfirmed potholes submitted by multiple users. Crucially, each identified Pothole has a stored “reports” value that keeps track of the number of reports. Filters can be set on fetch that allow for filtering based on the number of reports, allowing for filtering out potential unreliable reports. The specifics of this implementation will be explored when examining the entity-relation diagram.

Once the layout of the database is established, an API must be implemented to interact with it and perform the requested actions on those resources. This API will have endpoints for the frontends to ping when it detects a pothole and to upload an image of the detected pothole. Changes made in this department include both creating a new web frontend, and

modifying the desktop app to directly leverage the new database and API. The primary of these data visualization tools will include a map of pothole detections, which users will be able to browse and examine in detail. The team believes providing this functionality increases the potential use-cases of the system, as commuters can view where on their routes they can expect to find the most potholes and municipalities have more control over dispatching repair teams.

With all the goals for the backend laid out it should be expressed as to why these were stretch goals in the first place. There are a number of reasons for this, one of which is to ensure a robust foundation before moving into building any system on top of it. As explored previously, designing AI models and configuring them for field use is a difficult process. It can be months of fine tuning, retraining, and configuration before a trained AI model can be considered “complete”. This can be a labor intensive process, requiring multiple team members and may potentially halt development entirely if the training of the model is unsuccessful. The team does not wish to cut corners in the AI training process, and assigning these goals as “stretch” is a way to ensure that. The core project is around creating an AI model to detect potholes, and establishing an app that can use it on smartphones. Achieving this goal ensures that this team’s project has accomplished something meaningful, and should be the utmost priority during development. That does not mean greater value is placed on the AI over the secondary applications, it simply means the team recognizes that this must be treated as a project with discrete sequential parts, and treat it as such.

The team is confident that these stretch goals will be implemented in some form. Ideally, both will be implemented in their entirety, but as reiterated above, this is secondary to the primary goal of developing an AI to detect potholes and an app to run it. Despite this, to ensure the possibility of

developing these stretch goals the team had a plan for transitioning the team to develop the stretch goals once the initial AI and mobile application have been developed.

On the mobile team, they will be free to work on other things once the initial app is developed. Since the model will likely be the main item receiving sporadic updates, the mobile team may be able to downsize or transition to incorporating the stretch goal functionality into the mobile app. This may take the form of most members of the mobile team transitioning to API and Database development, as those will be required before the mobile team can begin integrating functionality.

The next sections will explore the stretch goals in depth, examining their technical requirements, challenges, and the plan for implementing these systems.

15.2 Database Overview

The most important component of the stretch goals is the database. The database is the foundation for the entire stretch goal project, enabling the group to permanently store potholes detected by the AI. This will take the form of a PostgreSQL database hosted by Supabase, for reasons which will be explained below. The PostgreSQL database will manage several tables, primarily one for “user-submitted” potholes, and another for “verified” potholes. A relationship between these two tables exists, where “verified” potholes have many user-submitted potholes, and user-submitted potholes are associated with one verified pothole. The information stored in each table will contain locational data tying the pothole report to a real-world coordinate, which can be used for assembling verified potholes or associating verified potholes with coordinates.

The primary purpose of the database is to store pothole information associated with real-world geographic coordinates, so the database can validate and report accurate pothole information. The database will be dealing with a large amount of user-generated pothole reports, most of which will likely be duplicate reports. Providing the infrastructure for consolidating these user-generated reports into a source of truth .

is essential to creating a database with some form of meaningful data. In this section, there is less concern with the “how” of accomplishing this discrimination, but with providing the necessary database resources to make the process simple and clean.

It also should be noted that creating this database can be seen as a substantial accomplishment, as simply having tabulated information of pothole reports can be very valuable to existing public entities for their own dissection. AI models could be trained to identify underlying causes for potholes using this dataset, or be processed statistically to generate insights into where potholes form, and how long they remain unrepaired. Creating the infrastructure in supporting this data collection is an essential first step in making the extended goals of this project a reality.

15.3 Supabase

A recurring provider that will be used in the stretch goals is Supabase. Supabase is a “Backend-as-a-Service” provider that specializes in simplifying backend functionality by providing a premade authentication system, configured database, and intuitive dashboard. Supabase is an open-source project, and can be self hosted using traditional means, or managed by Supabase for a cost of $20, or for free with restrictions. Supabase was

founded in 2020 as an alternative to Google’s Firebase, with a few key differences explored later in this section.

There are several reasons the team is choosing to utilize Supabase for this project. A major one is ease-of-use. Supabase manages all hosting for the database, which is preferred over manually setup and management. Managing deployments can be a hassle and failures may lead to a catastrophic loss of data. It would be most wise to offload these concerns when possible, so allowing Supabase to manage hosting of the database is a good choice. Supabase offers a generous free tier, allowing unlimited API calls and 500MB of database space. This should allow the group to get up and running on a proof-of-concept scale without needing to purchase a membership. Once scale is reached, it may be wise to host with them. For $20 a month, they offer 8GB of database space and support for 100,000 daily active users. Anything beyond that number of daily active users is beyond the scope of this document.

Another useful feature of Supabase is that it has a fully featured authentication system. Authentication for the system will be discussed further in the document, but authentication for the system will require some level of database privilege. There will be a separate process that will trigger certain API requests to happen at certain time intervals, and this will require elevated privilege compared to anonymous users. Supabase can handle this with relative ease, and works alongside database-level securities outlined in the PostgreSQL section of this document. As mentioned previously, securing data is of the utmost concern, and Supabase is a cost-effective way to guarantee that. With large-scale authentication schemes unnecessary for the current scope of the project, Supabase has more than enough out-of-the-box authentication functionality for the project. A recurring theme in the technology choices in the stretch goals is leveraging existing technology to make their lives easier. By offloading the need to build a

dedicated authentication system,there can be more time to perfect the parts of the project that add the most value. The team did not end up using this in the final product, but offers an avenue for improvement in the future if necessary.

Supabase is open-source. The underlying Supabase system is free to host on one's own, without the need to host with Supabase directly. While there is no current plan to do this, it does mean the system is largely modular, and can be deployed separately to any partner. It also ensures all the technology used for this project is open-source, keeping the project free from licensing issues or its own re-use, as the project is open-source itself. This makes Supabase a good choice for an open-source project like ours.

Supabase is designed to make things easy when building systems, and comes with several client libraries. The team will likely be using both the JavaScript/TypeScript client library and the Flutter client library, for the web API/app and the mobile app respectively. Supabase offers additional features for TypeScript development, including the generation of types for the database for easy access. Supabase is a good choice because it plays well with the technology stack being used for the stretch goals, along with the previously developed mobile app.

Before deciding to use Supabase, the team explored other options. The most obvious choice besides Supabase is Firebase, Google's Backend-as-a-Service (BaaS) project. Firebase is very similar to Supabase in the sense they are both BaaS providers and work to offload the same development concerns. Both offer fully-featured authentication systems, databases, and interoperability with JavaScript/TypeScript and Flutter. Firebase and Supabase differ in a few key ways. The first is that Firebase is not open-source, and Supabase is. This makes Supabase more appealing for the project compared to Firebase, as AI Pothole Detection is an open-source

project as well. Additionally, Supabase offers a more generous free tier while Firebase charges mostly for database reads and writes. This too makes Supabase a more appealing option, as the project is largely a price-adverse group due to the lack of sponsorship.

Supabase's choice of PostgreSQL as its underlying database brings the reliability, robustness, and features of a mature relational database management system (RDBMS). PostgreSQL is known for its ACID compliance, which ensures data consistency, and its support for complex queries and transactions. This makes Supabase suitable for applications that require strong data integrity and relationships between different entities.

On the other hand, Firebase's Firestore employs a NoSQL approach, specifically a document-oriented model. This provides a flexible schema, allowing developers to store and retrieve data in a format that suits their application's needs. Firestore's scalability is noteworthy, making it well-suited for applications with rapidly changing or unpredictable data structures. However, the lack of a predefined schema might lead to challenges in maintaining data consistency and relationships compared to a traditional relational database like PostgreSQL.

While choosing between the two is largely a matter of personal preference in regard to the concerns and features mentioned above, the team is opting with PostgreSQL as the database choice for its row-level-security functionality and robustness. The next section about PostgreSQL will highlight some of the features the team wishes to take advantage of, including row-level-security (RLS) policies that are unique to PostgreSQL.

After considering the above information, the project is planning on using Supabase and its PostgreSQL database to address the database requirements. Supabase is a reliable third party for essential backend needs,

and plays well with existing technology. Supabase handles hosting and authentication, two concerns that we'd rather not implement custom solutions in the interest of time-management. While most of these features are offered by Supabases' competitor Firebase, Supabase offers a more generous pricing scheme and is open-source, allowing hosting elsewhere if needed.

15.4 Original Database Plan

PostgreSQL is a popular SQL relational database. It's an industry favorite for data-centric projects, and provides a much greater level of extendibility compared to contemporaries like MySQL. The extendable nature of PostgreSQL will come in handy when tooling it for location based queries, which will be elaborated on later in this section. Besides PostgreSQL's extendability, it also sports a killer feature in the form of RLS queries. RLS stands for row-level-security and allows the group to set specific rules on what kind of rows can be inserted, updated, deleted, and selected in a given table, greatly enhancing security by adding another layer of protection.

RLS is a way to fine tune exactly what rows in a table a given user can access. This would have aided the application in several ways. To illustrate this, let's remind ourselves of the fact that the original system would have some level of permissions to handle having users being able to see unverified potholes, but simultaneously be allowed to create them. This directly translates into RLS policy, where SELECT operations would be restricted for anonymous users, but INSERT would be allowed. The team can very simply and very quickly write a SQL conditional statement that prevents any user from reading from the table, while another RLS policy allows insertions into the table. RLS allows the user to independently set conditions

for insert, update, select, and delete, without worry that the user will accidentally forget a check in the API and allow regular users to post.

Once RLS policies are enabled for a given table, all actions are restricted. By default, RLS prevents all database operations unless expressly told otherwise. To implement a user that would have elevated permissions compared to the anonymous user, i.e. a server that would analyze unverified potholes, the user would have the server sign in to Supabase and write RLS queries to enable those with the “admin” role to see and create for that table’s rows. This provides database level data protections with very little effort. This means that if an error were to occur in the API that allowed requests to be triggered without proper checks, they will still fail if the RLS policies do not allow them access to the table. While we did not implement, these features,

Row Level Security policies means the mobile app can actually just use a single anon key to contact Supabase. This would have come in handy when developing on mobile and the SvelteKit project. Since all actions are restricted based on RLS policies, the database will throw an error if a given user is outside their permissions. All of this greatly simplifies development complexity, and would have prevented the project from getting bogged down in the tedious creation of a robust authentication process.

A question that hasn’t been answered up to this point is how exactly the database tables would have been set up. The database would have followed the traditional resource-oriented database configuration, where each table in the relational database represents a group of a given resource. Creating a system where those responsibilities can be added without

compromising the core functionality was a primary concern.

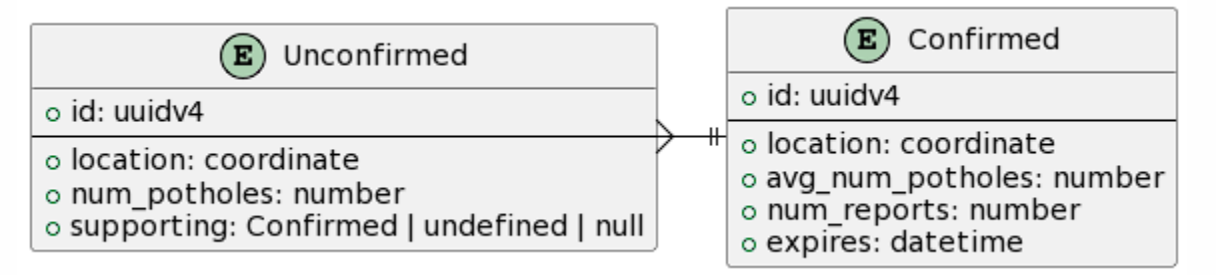


Figure 15.1: a hypothetical ERD of the Pothole Database

At the core of the old system, there are two entities: “confirmed” and “unconfirmed”. Unconfirmed potholes are representative entities for unconfirmed pothole reports from the field. Confirmed potholes on the other hand, are not directly the result of field reporting, and are extrapolated from the Unconfirmed resource table. The specifics of this extrapolation will be explored later.

The fact that ConfirmedPotholes are derivatives of UserSubmittedPotholes illustrates a relationship that can be represented in an Entity-Relationship Diagram (ERD). ConfirmedPotholes have a one-to-many relationship with UserSubmittedPotholes, where a ConfirmedPothole is fundamentally a collection of UserSubmittedPotholes. An inverse relationship exists where UserSubmittedPotholes only belong to a single ConfirmedPothole (or none).

The relationships between these two entities encapsulates the purpose of the database. The database is preparing unverified pothole information and consolidating that information into “confirmed” reports. The database will keep track of how many “unconfirmed” reports substantiate a “confirmed” report and use that data to determine how long until the database cannot consider this report ”confirmed”.

Going forward, the representation in the database for the unconfirmed potholes submitted by users will be referred to as “unconfirmed reports” and publicly visible confirmed potholes as “confirmed reports”.

One of the key differences between unconfirmed and confirmed reports is how they are created. Unconfirmed reports are submitted by anonymous users of the AI Pothole Detection mobile app, while confirmed reports are generated internally using an internal set of statistical heuristics.

Another key difference between unconfirmed and confirmed reports is who they are visible to. While unconfirmed reports can be *submitted* by anyone, they *cannot be seen* by any public or anon user. Unconfirmed reports are only visible to the internal systems that operate on “confirmed” potholes, i.e. any routine that needs to determine when to create, update, or delete a confirmed pothole resource. On the other hand, confirmed potholes are explicitly visible to the public, and are representative of what the team would consider a collection of ongoing reliable reports of a pothole.

In the next section, the implementation of these restrictions practically on the database layer will be examined, providing an extra layer of protection outside of the API layer.

Another thing to notice regarding the setup of the PostgreSQL database is the type of data estimated to be stored in the tables. As mentioned above, the database has two distinct entities: a “unconfirmed” pothole entity, and a “confirmed” pothole entity. These two entities share much in common, and that extends to the data they store. At a high level, both share two essential things: associated geolocation and number of potholes detected. These two identifiers represent the core information the system wishes to know about a pothole: where it is and how many.

The “confirmed” and “unconfirmed” resources are defined by their differences, namely with the “confirmed” resource containing extra data facilitating data management. The Confirmed resource has a ‘avg_num_potholes’ instead of a ‘num_potholes’ to show that this resource is aggregating the values of another resource. There are two completely unique properties to the Confirmed resource. One is the number of reports made supporting this confirmed report, which is to say the number of Unconfirmed reports that are “supporting” this Confirmed resource. The other unique resource is the ‘expires’ datetime property, signifying when this Confirmed report should be marked as ‘stale’ or possibly deleted.

The database is an essential piece of infrastructure for accomplishing what the stretch goals have set out to do. It provides the foundation for the rest of the project, and while this will be the first component of the stretch goals to be set up, it must reflect what is to come later on in the project. The hope is that by planning ahead now, there will be a smooth setup of database operations in preparation for building an API.

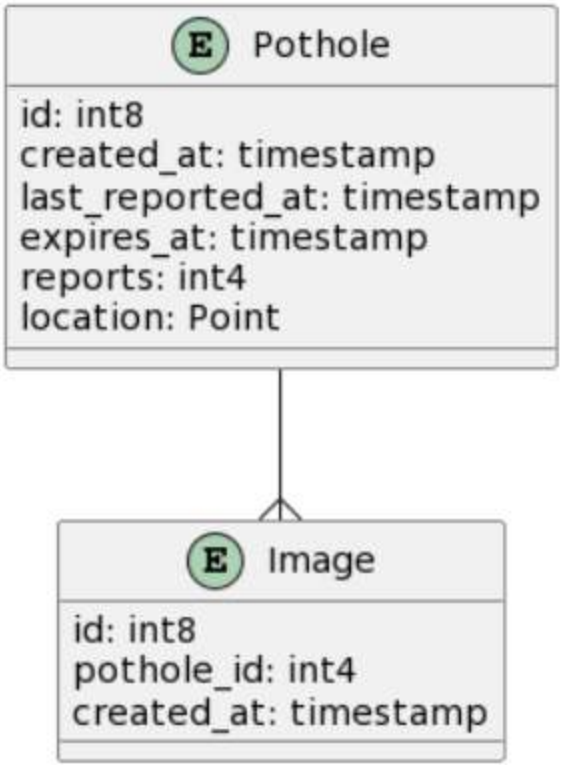
The team plans to make use of PostgreSQL’s Row-Level-Security features to allow some parts of the database to be available to the public and others unavailable to the public. As gone over extensively in previous sections, the database revolves around a primary relationship between two core resources: the Confirmed and Unconfirmed resources. Confirmed resources are linked to many Unconfirmed resources, which are used to calculate its own properties. This establishes a one-to-many relationship, where one Confirmed resource can have many Unconfirmed resources deciding its stored property values. The team never wants to expose Unconfirmed resources, as their information is by definition *unconfirmed*, but the database should allow any anonymous user to view *confirmed* potholes. This will be accomplished through Row-Level-Security (RLS) policies

Row-Level-Security policies are database-layer protections that prevent unauthorized database operations, whether that be inserting, updating, deleting, or selecting. They take the form of conditional SQL queries that are run against every database request, modifying the information returned. Several RLS policies will need to be written in order to secure data.

Regarding the Unconfirmed database, the database will need to create a RLS policy that ensures that anonymous users may insert into the Unconfirmed table. The database would want to create RLS policies that would restrict the update, delete, and modify operations on this table, as they will not be modified or deleted by standard requests. On Confirmed tables, the database wants to restrict all actions except for select for anonymous users. Confirmed resources are visible to anyone, but only modifiable by internal systems. The team should add RLS policies that only enable create, update, and delete operations for an authenticated server. This can be implemented through standard authentication procedures with Supabase, and will be examined in depth later.

Beyond the database lies the API, which will be the preferred way users will interact with the database. The API will be created using SvelteKit, which allows for the creation of robust REST APIs. The mobile app will contact this API to relay unverified pothole data, and the API will be in charge of storing this information properly, being some kind of CRON job that would run through the database and upgrade any collection of unverified reports that meet certain thresholds to visible verified reports. These verified reports would be exposed to users through the API, allowing for use in the web-app portion of the project.

15.5 Final Database Structure



The final database structure deviated from the original plan. The database surrounds two major resources: Potholes and Images. Potholes represent a unique report of a real life pothole. Images are images of real life potholes, submitted separately to potholes. Pothole and Image resources have a one-to-many relationship, meaning one pothole may have many images of it in the database.

Shown is an image of the current entity-relationship diagram (ERD) of the project. Images are crucially stored separately and associated via ID to the potholes to establish the desired one-to-many relationship that we desire.

This shares many similarities with the old model, but is much simpler. Additionally, expiry is stored on the Pothole resource themselves and can be used to filter out invalid results.

15.6 Final API Design

It is best to first start by describing the final web API design before delving into the original plan. The final API was an Express.js application that accepted and served JSON. It followed traditional REST standards in terms of naming and its resource-focused design.

The API was developed using Bun, which is a drop in replacement for both Node and NPM (Node Package Manager). While the runtime it offers is

often faster than Node.js, the primary reason we chose the runtime/package manager is its out-of-the-box TypeScript support.

TypeScript adds static type checking and annotations on top of JavaScript. It adds many of the benefits of statically typed languages with the ergonomics of JavaScript.

The following endpoints were created:

- **GET** /potholes?minLat=<>& ... &maxLong=<>
    - get potholes in bounding box
- **POST** /potholes:report
    - report detected pothole
- **DELETE** /potholes/[id]
    - delete reported pothole by id
- **GET** /images/[id]
    - get image from id
- **GET** /images?pothole=<>
    - get images of specific pothole
- **POST** /images
    - submit image of a pothole

These endpoints were used to create the functionality in the frontend and backend. The API checks during the report endpoint for an existing Pothole resource within 150 meters.

15.7 Old API Design

The web API will be the primary gateway towards an external entity interacting with the system. The frontend will make calls to HTTP endpoints using the standard HTTP verbs which will perform actions or yield results depending on the request made. Primarily, this will take the shape of manipulating two resources in the database by performing requests on them.

The API must have endpoints suitable for every type of resource-centric request a user could make. This includes traditional user use cases, like reporting an unconfirmed pothole, and niche server requests like pruning and managing confirmed potholes. Both of these kinds of requests must be accommodated by the API, and incorporate proper checks to give responsive feedback on the API.

A goal of this API is to be resource oriented. A resource oriented API is one that is centered around the creation and management of resources. (Resource-Oriented Design | Cloud APIs, n.d.). In previous sections, it has been outlined what constitutes a resource in the system. A resource in this context refers to a discrete set of data, usually representing something. In this case, the database has Unconfirmed and Confirmed resources, representing confirmed and unconfirmed potholes. Unconfirmed potholes are potholes reported by a user, meaning the Unconfirmed resource table in the database could have multiple rows referring to the same set of potholes. Confirmed resources aim to be an accurate description of a group of potholes with an associated area, and not have duplicate rows describing the same pothole/set of potholes.

Confirmed resources are created as a result of an Unconfirmed resource being created. Before an Unconfirmed resource is created, the API will check for a Confirmed resource in the same reported area. If there is one, the Unconfirmed resource will be created “supporting” the found Confirmed resource, also updating the found Confirmed resource with the given stats.

The team also would like an endpoint to trigger a database pruning, where the database removes all the “stale” reports. A “stale” or expired resource is a Completed resource that has not been updated in a recent time frame. It can be assumed this means the pothole has been filled or has

otherwise been repaired. This pruning process will remove the entries in which the expired property has passed the current date. This endpoint will likely be triggered regularly through the use of CRON jobs or serverless functions. This function will require elevated permissions, a subject that has been elaborated on in previous sections. This process will be explained in detail further in this section.

The purpose of the API is to allow frontend interfaces to communicate with the backend to support the creation of pothole reports. It is good to keep in mind that the database does not need to take into account grouping of potholes or any sort of logic to determine where they were detected, as that will be handled inside of the mobile application code. The team is solely concerned here with the management of resources, and how this app will generate and maintain them.

First, the team will take a look at how it is intended to take in requests from the mobile app indicating that they have located a pothole (or collection of potholes) at a given location. This will take the form of a POST request to the "/api/pothole" endpoint. This endpoint will be responsible for producing new Unconfirmed resources, representing the users unconfirmed report. The request will expect a json body containing the location of the pothole, the number of potholes reported, and a signature. Once this request is received on the server, it will process the request by first checking if there is a Confirmed report in the same area that this request was made. If there is, the API will assume these reports are talking about the same pothole, and associate this report with the confirmed report. If there is not, a new Confirmed resource will be made, with the newly created Unconfirmed resource referencing it. An important fact here is that the Unconfirmed report is created after identifying or creating the Confirmed report it will produce.

It should be noted that not every Confirmed report may be shown to users in the final stretch goal interface. The API will likely set some form of threshold restricting shown Confirmed resources to those with a certain number of substantiating reports. This will be explored further when discussing the implementation of the map interfaces.

Since the creation of Unconfirmed resources will trigger the creation of a Confirmed resource,that process will be gone over next. Creating a Confirmed resource is a privileged action in the system, and requires elevated privilege. For this, the server will log in to the system using authenticated credentials, and perform the request. This will follow a similar pattern as to a Confirmed report, minus the need for cryptographic signatures. This request will be made to a "/api/potholes/confirmed" POST endpoint with a request body matching the required fields needed for the database.

The team will also need the ability to modify Confirmed resources once they detect additional reports. It's important to be able to increment the number of reports, as well as update location if needed, as well as add a certain number of weeks/days to its expiration date. To accomplish this, a partial update API design pattern, as laid out by JJ Geewax in his book *API Design Patterns* (Geewax, 2021), will be used. This involves creating a single endpoint, where the column the user wishes to update is included, along with its value. This would look something like "/api/potholes/confirmed/fieldMask=[field]", where a PATCH request would be made to this endpoint url with the "[field]" would be replaced by the desired field. This pattern ensures that two users updating different parts of the same resource do not conflict with one another. The process is very similar to the other ones, except as this is a privileged action that does not require the cryptographic signature. Instead, the group can simply extract

the new parameter value from the body and examine the url to figure out what is being updated.

In order to perform reads of the “confirmed” pothole database, the group must implement some paginated search endpoints. These endpoints will need to be established for the Confirmed resource specifically, and permit searching by multiple parameters. Users of the map and dataset will want to see the reports in a given variable radius around a variable location. This means the group will need multiple paginated endpoints for the Confirmed resource.

The way the group will set up paginated endpoints will be based on the paginated endpoint pattern described in API Design Patterns by JJ Geewax. This endpoint is completed by two separate urls:

- “/api/[resource]/pageSize=[size]&sortBy=[parameter]”
- “/api/[resource]/pageSize=[size]&sortBy=[parameter]&pageToken=[token]”.

The first url is to get the first page of results with a maximum of size items. After the first request, the user is given a “page token”, a token in the second endpoint url to get subsequent pages of results. Each request returns a page token, with its value either being a base-64 encoded string or null if there are no further items. In this implementation, this base-64 encoded string will likely be the last index included in the search, to be decoded on the next run and be used to calculate the next starting point.

At first, this pattern may seem odd. Why not use the index of the final item as the start of the next and include that? Why encode the string at all? The point of this is to obfuscate how the API calculates its next page. If the index is used, it may limit the options to query the data. While this likely will

not happen, the group opted for this pattern as it is battle-tested and industry standard, and very unlikely to lead to complications.

The first paginated search endpoint that will be required is an endpoint to get a list of Confirmed resources sorted by distance to a given location. This endpoint url will look something like

- “/api/potholes/confirmed/pageSize=[size]&sortBy=[location]”
- “/api/potholes/confirmed/pageSize=[size]&sortBy=[location]&pageToken=[token]”

The implementation of the actual endpoint in code will leverage Supabase’s included client library, which includes methods for getting paginated results from the endpoint. The pageToken will be converted into a starting index for Supabase’s pagination function, and the results will be formatted into responses.

Supabase offers a fully featured authentication system, complete with login, registration, email confirmation, etc. The Supabase client must be signed into client-side, and this client-side instantiation will be passed into the backend for use in the API. The specifics of this will be given in the SvelteKit section later on in this section.

The Supabase client by default is loaded with anonymous permissions, meaning these are the default permissions provided. This means there is no id for the current user in the database, and cannot use anything to determine what permissions this anonymous user should have. The default permissions of the average user must be created. To set up the server-tier permissions, there will be a user using the built in Supabase sign up feature, but once it is done being used registration will be disabled. This will be the only “user” in the system with authenticated permissions, and can be used by RLS and Supabase to give permissions. The group can incorporate this

into RLS queries directly. While not necessarily the intended use of its system, this would work as long as the team does not plan to incorporate accounts into the authentication schema. Even then, with modifications and expansion of a most robust authentication system, this option will still work.

One downside of using a regular account for elevated server privileges is that it requires a less elegant implementation. Since the privileged requests happen from inside an API request, the API must be able to have a Supabase session with the default permissions loaded alongside the user's anonymous Supabase session. This is not a major issue, and does not impact performance in any way. Instead, it is a less elegant solution that may produce more confusing code further down the line.

Previously mentioned, while building the API, validation checks for permissions will still be included even though RLS would prevent the operation from occurring. The reason for this is creating a clean API. When other subteams of this team are using the API, it would be much more intuitive for them to receive a detailed error message during development, rather than a generic RLS error that gives very little information on what went wrong.

Another reason is that certain operations that RLS prevents will return empty data, instead of an error. This could be confusing. Without code checks for permissions, the recipient could receive empty data which may be interpreted as just no results. Providing checks in code allows the group to return rich error information to the users of the API, making development easier for all of us.

15.8 Conclusions

In the above sections, the database was laid out for the project. As iterated in the previous sections, The group would configure the database to manage two primary resources: Unconfirmed and Confirmed resources. Confirmed resources represent “confirmed” reports of potholes, and are exposed to users through the API and Unconfirmed resources are uploaded to the database through the API by anonymous users using the app. Configuring the database to handle these resources securely means creating row-level-security (RLS) policies, a feature of PostgreSQL, the database used by the chosen provider Supabase.

Our API will be written using SvelteKit in TypeScript, and will have several endpoints for creating and querying the data. Paginated endpoints will exist allowing location based search in a given radius around a given location, for use in the frontend map and warning systems.

Once the API is created, all that is left is to create a web app containing a map of pothole data and upgrade the mobile app to have enhanced early-warning capabilities. This is to make the app more valuable, and show off the useful data that has been collected. The group believes by showing off this data, it can grow and entice users to use the app and service.

Creating a database of pothole information is one of the greatest potential benefits and use cases of the AI model and app. This information could be used by local communities and governments to improve road quality and identify areas with recurrent problems.

16 Front-End Web Application

16.1 Web Application Overview

The ultimate goal of the stretch goals is to have a public interface where users can browse the collected pothole data in an intuitive and informative way. The primary way the group wishes to accomplish this is with an interactive map where users can see exactly where around them have the most potholes.

This goal would be conducted in two concurrently running parts, one for the desktop application and one for the web. The web app portion will likely be the most labor costly portion, as a frontend web interface must be included. This frontend web interface will only be used for viewing pothole information as well as generating reports of an area of potholes. The main use of the web application would be the public who is looking to find information on the locations of potholes. The desktop application has additional features where a user could add or delete a pothole from the map, this is because the user base for the desktop application will comprise mostly of municipality workers.

The web app would be composed of several components, and make use of the Google Maps API for displaying a map with annotated markers. Users are able to manipulate this map, and it would update to display the reported potholes on screen. Users can then select a pothole, and view information about that specific pothole. Users are also able to generate a report of all the potholes that are currently in the view of the map and download it as a csv.

16.2 Web Application Architecture

Designing web application architecture is the foundation of a successful web infrastructure system. There are many languages and frameworks to choose from, each serving their own purposes with advantages and disadvantages. Due to the simplicity of our web application, the team decided not to use any specific framework to develop the client side portion of the application. The reasoning for that is due to the fact that the team's system lacked the need for numerous pages that needed to be routed. The overall goal of our client application was to simply display a map of potholes for anyone to view. A framework was considered for the use of having an instant DOM refreshing feature without the need to refresh the page but it was later discovered that was achievable with just regular javascript. With all of that being said, our client side web architecture consists of HTML, CSS, and BootStrap for styling as well as javascript for handling our http requests to necessary APIs.

16.3 Web Application Style

When it comes to web application styling, the team wanted to keep it simple. It was finally decided to use BootStrap5 to help style the website due to its powerful out-of-the box features. BootStrap5 will allow anyone to style HTML elements with predetermined colors, sizing, and spacing options all by assigning any HTML element to a specific bootstrap class. Not to mention it also allows for a web page to be fully responsive with this method. Those reasons along with the quick start up ability and easy to use framework of BootStrap, are why the team decided to use it for styling. Aside from BootStrap, the team also had a global CSS file within the web application to handle any specific HTMl element that needed enhancements beyond BootStrap's abilities.

Taking a look at Figure 16.3.1 below, this is the fully complete web application. With the name of our project at the top, the export to CSV button, and the map there are no other elements in the site. Google Maps API also has a couple different settings for the map. The main one being able to change from the standard map view to a satellite view as well as adjust the zoom and change the view position of the map.

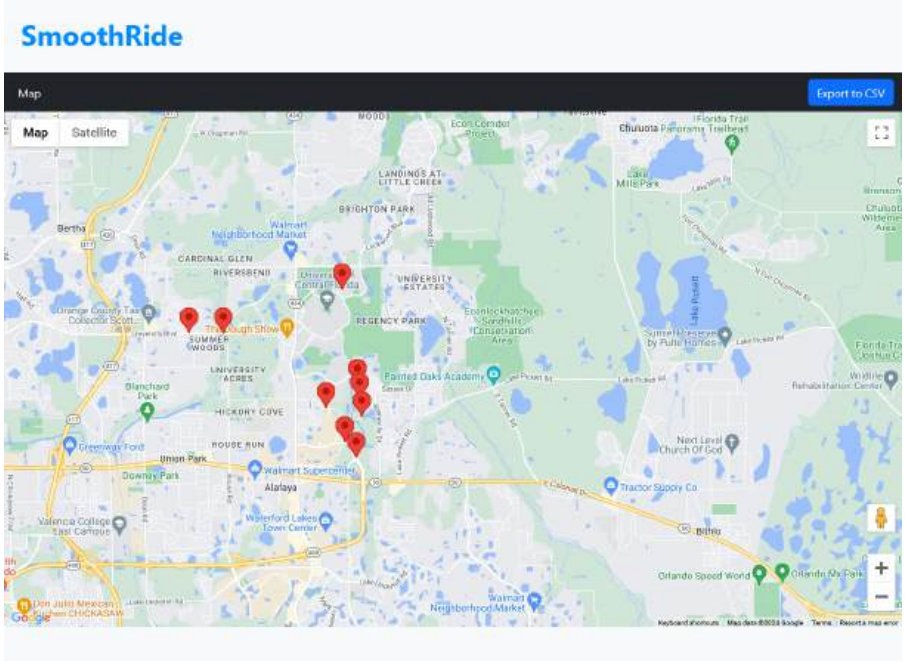


Figure 16.3.1: SmoothRide Web Application

16.4 Web Application Maps API

Google allows fairly open access to their Maps API which allows users to generate their own custom maps loaded with any custom markers that are necessary. This was exactly what our group needed for this web application and the best part is that fact that it is free and comes with its own settings to help restrict the API key.

Firstly, our group had to create a Google Cloud account and apply to obtain an API key. This did not consist of anything more than creating a billing account as well as a few captchas and questions about how the key would be used (personally, or commercially). After obtaining the API key, Google Cloud makes it very intuitive and simple to restrict the key in various ways. The first option we opted for was to restrict the key to only be allowed access to the Maps JavaScript API as we would not be using any other APIs or SDKs. The second restriction we opted for was the website option, which only allows use of the API key on specified websites that we own.

Now that the API has been obtained and restricted in order to protect it, we were able to start creating our map. Also in Google Cloud, we have to create a custom map that will be used in tandem with our API key on the web application. The map we selected was based in Javascript as that is what we were using to create this web application and the map was implemented with Raster over Vector, because Raster is supported by all platforms as Vector is more new and not supported. After inputting these settings Google Cloud will generate you a unique Map ID that will be used in the API call to generate the map.

Google Cloud also gives plenty of customization options when it comes to the map that you are creating. With these customization options we could choose the theme of the map as well as the style and if necessary an entirely custom environment can be made to make the map look however the user decides. We did like the idea of having a custom map but all in all we found it was unnecessary with the time that we had and chose to just use a default map.

After all of this is complete we can now implement a map into our web application. There are plenty of very in depth articles about how to use the Maps API that made the base implementation a smooth process. After this

was done we now were able to start adding in our custom markers for the potholes.

Custom markers are created and placed on the map using javascript code. It starts with the initMap function that will initialize the map. Within that function there is an event listener that will trigger whenever the location of the map changes, as in when a user will scroll to a different area on the map. When that even is triggered, it will grab the coordinate values of each corner of the map and then send a GET request to the backend's API. The GET request will return an array of potholes that exist within the area of the four coordinates that were sent. Once that array of potholes is obtained, the function loops through it and places a marker at each coordinate location using the google.maps.Marker() function. This method allows the webpage to only ping the API for potholes within a specific area and it will actively do so which will limit the amount of data being sent through http requests on the client-server system.

16.5 Web Application Deployment

Once the web application was completely fleshed out we looked into a couple of options in terms of deployment. The method that stood out to us immediately was Amazon's AWS Amplify. This was a very simple and easy to use deployment service that was also free to use. The entire process consisted of having the project folder of our web application and dropping it into Amplify and pressing the deploy button. After a few seconds it would generate the link for the website that was now live and open to access. We would then take that link and add it to the Maps API restrictions to ensure that everything was working correctly, and the web application was deployed. There was no option for customizing domains because only the free tier was used and we were limited on how many times a site could be accessed in a month or the free tier would expire. Thankfully we did not

come close to reaching this quota and were able to keep the spending budget at 0$.

17 Testing and Evaluation

17.1 Model Testing

The program went through unit testing after the AI model was implemented. The app had to work in a variety of different situations, and it had to properly identify potholes in many road environments. For the majority of these testings, the app was run in multiple different circumstances and areas.

Functional Testing

- The app had to be able to identify potholes with high accuracy.
- The training the group had done before provided the ability for the model to work in photos, but the model is not run on video in the training phase. The model was as effective when tested on video and on the road.

Compatibility Testing

- The program had to work across various different devices. These products are available to be tested on. The model must work on both of the different operating systems.
- The program was run at multiple different times of day, and on multiple different roads.
    - The app is suboptimal when running at night time but it is still able to detect potholes. Because of the worse performance and

less coverage on potholes not hit by headlights, it is highly recommended to use the app during the day.

- The app will be run on roads that have no potholes to identify.
    - The app did not falsely identify potholes. The model parameters can be manually changed to require a higher degree of confidence in the pothole location. This data can be taken on many roads without potholes.

Integration Testing

- The model loads at the same time that the app does. It doesn't have a period at startup where it doesn’t work, so it’s well integrated
- The model was tested on detecting multiple potholes, and the implemented delay solves this issue.

Performance Testing

- The model doesn’t take up so much of the laptop’s GPU that it is laggy. There is no need to change the speed of the AI model, which we could if needed
- It was tested, and the model doesn’t have significantly load when it identifies a pothole. The model was not laggy, so the performance does not need to be changed. The app is intensive on the resources regardless, but it is manageable.

Robustness Testing

- It was tested if the model can identify a pothole when it’s very close to it. It can, but it’s less accurate. Luckily, it’s far more important for functional use that it can detect potholes when they’re very far away, however.

Usability Testing

- This will take user feedback on how the model performs when using the app as intended. We have been able to successfully utilize the app on the road live.

Internationalization Testing

- The model works on different roads in different countries, and doesn't misidentify much noise from these countries to be potholes. The model also has a lane fitting function where only the potholes in the lane can actually notify the user and be uploaded to the database
- This was tested using video data of other countries sent into the app. We did not fly to different countries, as that wasn't feasible.

17.2 Application Testing

The base application is rather simple so there is not a lot of testing other than testing the model inside of the application which is talked about in depth later in the Integrated Testing section. Earlier in the paper the different aspects of the program were explained in depth, and they will be touched on again here for ease of explanation, for more in depth explanations on each screen please see 10.0 Mobile Design and User Experience.

Every feature of the laptop program needs to work correctly. The map needs to consistently come up with all of the potholes and load as each section of the map is seen, but it's irrelevant for it to load potholes that are not in view. Deletion needs to delete the potholes not only from the app but also the database, and adding a pothole needs to add the pothole to the database's map, the website's map, and the map on the actual app. The

report needs to accurately show the potholes that were identified by the bounding box.

Another aspect of testing came with the lane detection software. When the lanes are enabled, we needed to double check that only the potholes identified in the lane were added to the database. The lane detection software also needed to be visible to the user, and it needed to be checked that the lane detection could be turned on and off, altered, changed from camera to camera, especially with different camera shapes. We found that the tests were always successful.

One screen to test was the main screen of the application, the "Camera Screen." This screen involves the application having access to the user's camera and for testing this functionality the camera's vision should be displayed across the entire screen. The navigation bar appears at the top of all of the screens that the application is currently planning on having. In order to test this the application should switch to the different screens as they are selected on the navigation bar. There was also a test to make sure that if the user is already present at the "Camera Screen," they will not reload the "Camera Screen" if they select it from the navigation bar. This removed any confusion as well as help with the efficiency of the application. There is no reason to reload a screen that is already loaded so there was a check for the user's current screen to ensure that this does not happen, which the application passed.

There was a lot of testing with the "Settings Screen." Inside the settings screen there is a slider that allows users to change the alert volume when a pothole is detected. This testing was done after the model was integrated but testing was also done before then to make sure that the application recognizes the slider as well as moves around when interacted with it. There is also an alert size slider that was tested in depth as well. For

now the same testing can be done to make sure the slider works and it is able to recognize when it is being interacted with, same as the volume slider. This navigation between the screens was tested to ensure the user has access to it.

The other testing that was done in regard to the base application had to do with different stretch goals. The first stretch goal that the group has for the application is creating an interactive tutorial for the user to learn more about how to use the application along with information the user can read to be informed about how the model works, but we did not end up creating this.

One stretch goal was the biggest one for the group and required the most testing at this stage. The final screen is a map of all of the potholes that user's applications have detected over time. This screen is connected to a database. This first testing done involves ensuring that the user is able to move from one screen to another. The map itself is interactable and allows the user to not only move around to see the area around them but also manually report potholes that either they know about or on the off chance the application fails to see for some reason. Testing was needed to be done to make sure the map is interactable, and the user is able to create reports and move around as they seemed fit. The largest group of testing involved the storage of the positions of the potholes and them correctly appearing on the map screen. This was crucial to make sure it is working properly because of the fact that some of the alerts rely on these positions and if the positions are incorrect, the application could falsely alert the user or fail to alert the user about the existence of a pothole.

18 Conclusion

The final results of the project are that we were able to train an AI model to perform pothole detection, a mobile application that was developed and later dropped, a laptop program that is set up to both perform object detection and allow for map controls, and a website front-end for viewing the potholes and generating reports. The AI model selected was able to achieve great performance on the pothole detection task, and was able to perform in a variety of road conditions, from dirt roads to wet to paved. The mobile application was able to run an AI model and generate alerts and database uploads in real time, but the model performance was not deemed acceptable. The laptop program did have acceptable performance results for both the model and the application itself, allowing for detection, uploading, and viewing the potholes on a map. The map allowed for the user to add, delete, and download a report on the found potholes, as well as view images of the potholes detected. There was a website developed that performed similar actions to the map in the laptop program, but only allowed for downloading of potholes. Finally, a database that records the potholes that have ever been collected, the number of times it has been seen, and when it should be removed from the database. This database automatically merges potholes that are near each other and removes potholes that are older than 6 months old, under the assumption that they are either patched or at least need to be checked again.

To summarize the project, AI Pothole Detection is a laptop and web application that uses YOLOv8, an object detection model, to detect upcoming potholes. This model is run by a laptop application that has three pages: The main viewing/camera page, a settings page for adjusting opacity and volume, and a map page for viewing the potholes on a map. The intended use of the program is for the camera to be used in the top center of a

driver’s windshield, similar to a dashcam. This will give the program, and thus the AI, a clear view of the road to maximize pothole detection performance. When a pothole is detected, its location will be estimated to be either on the left, right, or middle of the road. If the pothole is on the left or right, an associated directional alert will play from the program. The alert will both be visual and auditory, with the visual alert being performed by shading part of the screen and the auditory alarm starting which side the pothole is on. The map will display all the potholes collected and give the user control over adding, removing, and downloading a report on the potholes. The website is a public facing domain to allow regular citizens to view the database without modifying it.

The lessons learned in this project were on expectations, alternative plans, and testing. The team originally expected that the mobile application and the MobileNet model would suffice for operation, but after live testing and deployment during the first Leinecker demo, the team found out that the model did not work well at all. This significantly set back work, as the entire mobile application had to be scrapped and a new application made as quickly as possible. This taught the team a lesson on having back-up plans if components of the project fail, especially if some components of the project have a high risk of failure. The team also learned that they needed to have more robust testing. While there was some amount of actual live testing, it was not as significant as originally planned due to the extended development time of the model and the mobile application. This led to not enough deployed testing, leading to a subway demonstration with Leinecker. The team also had a lesson on deadlines. The team struggled to occasionally meet deadlines and schedule the required dates, such as the Leinecker demo and final committee. This was partly due to the head of the team not being in the Leinecker section and the team having scheduling conflicts, but this could have been prevented with earlier planning.

The future work that can be done for the application are fixing the mobile application, making the laptop program and website more robust, and making a more efficient AI model. The mobile application is still a well made program and was able to run the model, but the model did not perform well and there were slowdown issues. It may be possible to fix the app to use hardware acceleration better, and possibly integrate native code into the Flutter app to better utilize hardware resources without causing overheating. A new AI model could also be trained as well, or the MobileNet model could be improved. TensorFlow's Object Detection API does allow for quantization aware training, but that was only designed for TensorFlow 1 and not 2. It may be possible that using that system may lead to a better performing model that can be run on the mobile device. The laptop program and website could be made better as well. The laptop program is somewhat laggy and has multiple points where it can crash. It may be a good idea to rebuild the program in another Python GUI library or another language entirely. For the website, it would be good to have the same report functionality as in the map for the laptop program. It would also be a good idea to deploy the website to an actual URL.

19 Administration and Operations

19.1 Tools

Some essential tools that were used: Github, Jira, Visual Studio Code, RoboFlow Universe, Flutter, Android Studio, and Pytorch. Github and Jira were the first 2 tools that were setup by this group in order streamline work as well as setup organization with an agile development workflow. All of the group members are using visual studio code as their IDE because the group

has all used this the most and it provides a wide array of plugins that are required in order to complete this project.

The AI team has used multiple tools in order to obtain an entire dataset of images and start training test models. RoboFlow is used as the storage system for holding the entire dataset of images that the group is currently using to train the models. Ultralytics was also used to train models and will be used more throughout this project. There were multiple tools that were not listed above to train some test models as the group was getting to learn the AI aspect of the project more. Some of these tools include Google Collab and Pytorch. For training the final model the group does currently have the plan of training it on a local system but if the training is too intensive and takes too long the group will fall back into using the Newton Cluster that is provided to students by UCF.

The mobile team used Flutter to create the mobile application. For more information about this decision there is more information provided earlier in this document. Android studio was also used as well by the mobile team in order to create an application that is compatible for both IOS and Android. The team used a plugin known as Flutter Vision that allowed them to integrate the model into the application itself with the backup of using another PyTorch plugin that is compatible.

In the laptop program, the main tools that were used are Python, Ultralytics, Custom TKinter. Python was used as the base coding language for it, and Custom TKinter allowed for smooth integration of many of the features into the application. Ultralytics was utilized to run the AI model on the actual app itself.

19.2 Budget

Table 17.2.1 provides an itemized overview list of all the costs that will be and are incurred for the application on an annual basis.

Tool/Equipment	Purpose	Vendor	Total Cost
GitHub Organization (Free Tier)	Code hosting and collaborative management	GitHub	$0.00
Roboflow Universe Datasets	Image Dataset Storage (up to 10,000 images)	RoboFlow	$0.00
Flutter	Mobile application development	Flutter	$0.00
Jira	Team organization and agile development	Atlassian	$0.00
Visual Studio Code	IDE and text editor	Microsoft	$0.00
Android Studio	Mobile application development	Google	$0.00
Ultralytics	Model Training	Ultralytics	$0.00
Google Collab	Model Training	Google	$0.00

Newton Cluster	Model Training	UCF	$0.00
Local Machine	Model Training	Nicholas Gray	$0.00
PyTorch / TensorFlow	Model Development and mobile integration	Meta AI	$0.00
Total Cost			$0.00

19.3 Timeline

At the start of the project, the team had an initial broad estimate of when sections of the project would be completed. The initial estimate was that model training would be completed by the end of December and mobile development by the beginning of January, with a minimum viable product prepared for the beginning of Senior Design 2.

While operating under Agile, the timeline was set up for the first semester to give time for research, AI development, and mobile development to occur simultaneously and independently. The Sprint Timeline shown in Figure 19.3.1 below shows how the timeline was sectioned by Epics that broadly covered the development time necessary for covering the initial model and mobile developments. For the AI developments, the dates between October 18th and November 24th were set in order to find all the relevant information and utilize it in model development. This then connects with the Epic for Model Training, which covers from November 4th to December 12th.

Concurrently, mobile app development was given a longer development window from October 18th through to the end of December.

The timeline was structured in this fashion to give the mobile development team time to set up their environments, design the initial application, and prepare it for integration with the model.

After the model training, the mobile application continued up until the first demo, which is when the mobile application was scrapped and the laptop program was decided to be used. The laptop program was developed under Mobile App Enhancements and was started on March 26th and completed April 10th. The website was started on March 26th and completed on April 10th as well. The database started development on February 1st and continued development and modification through until April 10th. Live testing and integration happened through the entirety of Senior Design 2, starting from January 15th up until April 10th.

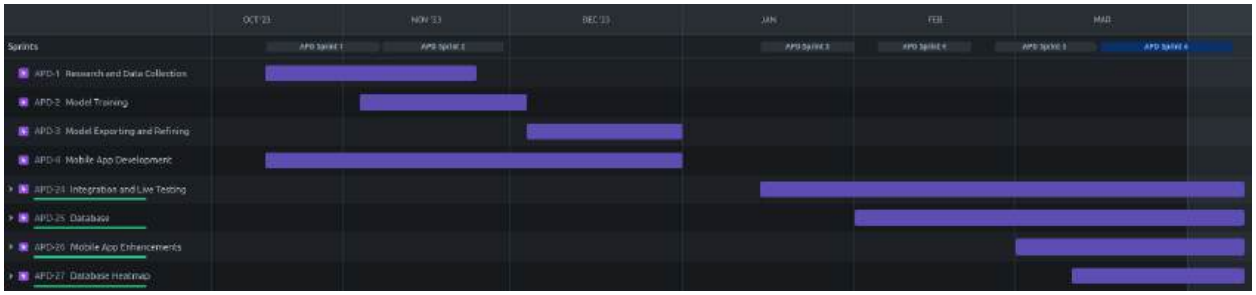


Figure 19.3.1: Jira Timeline of Senior Design 1 and 2

Figure 17.3.2 below is the specific milestone chart for the project goals as they were started and completed.

Task	Start Date	End Date
Decide on Model Format	October 18th	October 20th
90 Pages of Documentation	October 18th	October 27th

Initial Mobile App Creation	October 18th	November 8th
Data Collection	October 18th	November 11th
Data Uploading	November 12th	November 21st
Initial Model Training	November 24th	November 27th
180 Pages of Documentation	October 28th	November 29th
Integration and Live Testing	January 15th	April 10th
Database Development	February 1st	April 10th
Mobile App Enhancements	January 15th	March 26th
Laptop Program Development	March 26th	April 10th
Website / Database Map	March 26th	April 10th

Figure 17.3.2: Milestone Chart with Dates

19.4 DevOps

18.4a Version Control

Version control of the Flutter application is an essential part of efficient and smooth development. Without version control it would be very difficult

to produce and manage a working application effectively. The basic idea behind version control is when developing code, every significant change is decentralized from the main product and then re-inserted back into the main product once it is battle tested and working as intended. Every significant change is tracked and documented which allows anyone on the team to view the history of changes and roll back the version before any given change as needed if the system stops working due to new features or additions. While there technically are multiple different version control systems, there is one in particular that holds a huge monopoly over the version control service market and that is Git. Git is a version control system with a wide variety of command line tools, as well as interface tools but developers who do not plan on becoming baristas after their academic career choose to use the command line tools. These command line tools make tracking and documenting the application's much more streamline than any other manual process one may think of.

Before explaining the version control and development pipeline, it is important to know the different processes and commands of git, why they work, and what purpose they serve. Here is a list of git commands from the command line tool that the mobile team will utilize to perform correct agile development and deployment.

- git init - git init is the command used to initialize a github repository. This is the first command every developer uses when starting a project that uses version control.
- git branch - this command will list all of the existing local branches on the developer's local machine.
- git branch <branch name> - this command will create a new local branch on the developer's local machine. This new branch will branch off of the current branch that the developer is currently on. This is

used to create a new environment that the developer will use to add features to the codebase without tampering with the original branch during development.

- git checkout <branch name> - This command will switch the code base from the current branch to a new local branch on the developer's local machine.
- git status - this will list all of the untracked and tracked additions or modifications of the current branch that can be staged for a commit.
- git add <file name> - This command allows for the developer to stage any file listed in the <file name> section for a commit. The <file name> section could span anywhere from one file, to a list of files, a folder, or * for all files.
- git commit -m "<commit message>" - This command is used to commit all of the files that are staged, with a commit message relevant to the features added or changed in said commit.
- git pull - This command will update and sync the current local branch that the developer is working on with the most up to date version of its paired remote branch.
- git merge <branch name> - This command will merge the branch from the <branch name> field into the current branch that the developer is currently checked out into.
- git push - This command will push all of the staged commits from a local branch onto the remote branch. If this is the developer's first time pushing to the remote branch they must use "git branch –set-upstream origin <branch name>" to set the upstream remote branch to that local branch.

The version control deployment pipeline the group will use is known as a Continuous Integration and Continuous Deployment pipeline. Instead of

explaining how that works, it is more beneficial to discuss the development and merge process the team will use.

1. The main working product will be on the global Main branch, this branch will always have the most current working version of the application.
2. To add a feature, a developer will create a new local branch off of master and start making edits to adhere to the new feature. This Feature branch is also known as a Development branch or a Dev branch.
3. Once edits are made, the developer will test those edits on their dev branch and make changes accordingly until the dev branch functions properly.
4. Once the dev branch is tested, the developer will merge their dev branch into the Alpha branch. The Alpha branch is a branch that is a collection of every developer's Dev branch that consists of new features. Once a developer's Dev branch is on Alpha, they must test their new feature again to ensure it works in tandem with every other developer's new features as well. The Alpha branch is technically always ahead of the Main Branch because it has new features but it is usually unstable. Having an Alpha branch is how the group avoids running into major bugs on the Main branch, which should be a final working product.
5. Once the Alpha branch is tested and working properly, it is time to merge the Alpha branch into the Main branch to update the Main branch to a newer functioning version of the app.

That is a surface level explanation of the process and why each branch and step is important to keeping the Main branch at a continuously functioning state. Below this there will be a diagram of the process just explained, to

give a visual representation of how the CI/CD pipeline process works. Figure 17.4a.1 below displays the basic overview of creating a new feature branch from the main branch.

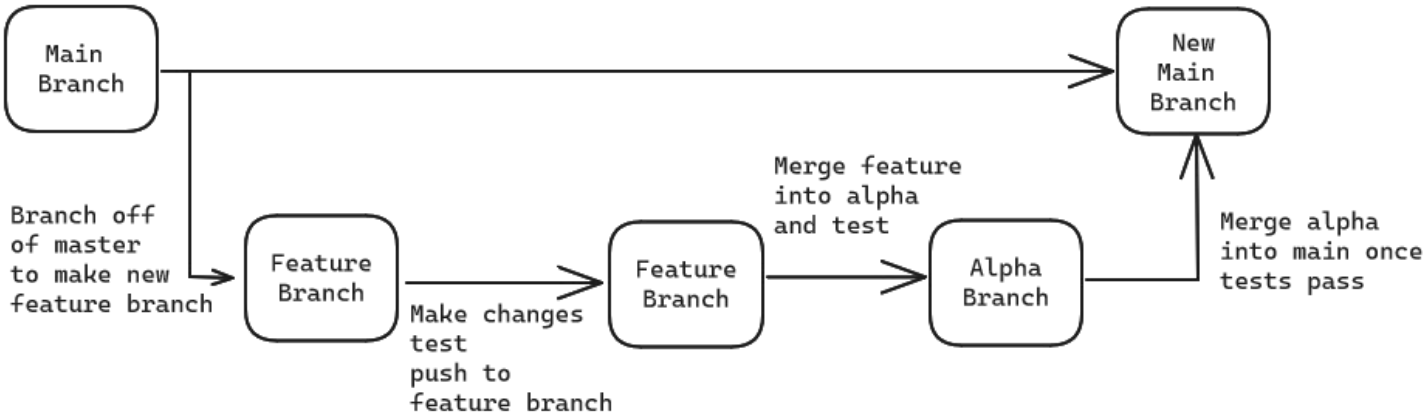


Figure 17.4a.1: Git Feature Branch Pipeline

19.5 Agile Format

The team operated using a custom form of Scrum with twice a week standups instead of daily standups, time limits of one hour for standups, and different tasks for each type of standup.

A single sprint would last for 3 weeks and would start and end on Wednesdays. Every Wednesday at 4:30 PM there would be a full team standup, where the entire team meets and updates the Jira board for completed and new tasks. On Saturdays and Sundays, the individual sub-teams would operate separate standups to update fellow team members on the current status of work before the next standup. These standups are looser than the full team stand ups and can be run at different times each week or run asynchronously. During standup meetings, each team member updates the project leader on their recent updates and if any stories need to

be moved. Once all team members have spoken, an action plan for the next week is made for what everyone will work on for the next week.

Meetings will have meeting notes recorded by one team member writing meeting minutes at every full team standup. These meeting minutes will be accessible by all team members and will be used to track conversation and updates and catch up teammates who missed any meetings.

Stories will be managed using a standard Kanban board with “Todo”, “In Progress”, “Review”, and “Done” columns. At the end of every sprint, the stories in the “Done” column will be deleted and new stories in the backlog will be moved into Todo as each teammate decides what they will focus on for the sprint. Each spring will have an overall goal, with the stories created and assigned relating to the completion of said spring goal.

20 Acknowledgements

20.1 References

American Society of Civil Engineers. (2021). *Road Infrastructure | ASCE's 2021 Infrastructure Report Card*. Infrastructure Report Card. Retrieved November 28, 2023, from https://infrastructurereportcard.org/cat-item/roads-infrastructure/

Arya, D., Maeda, H., Ghosh, S. K., Toshniwal, D., & Sekimoto, Y. (2022). RDD2022: A multi-national image dataset for automatic Road Damage Detection. *figshare*. https://figshare.com/articles/dataset/RDD2022_-_The_multi-national_Road_Damage_Dataset_released_through_CRDDC_2022/21431547

baeldung. (2023, June 17). *KISS Software Design Principle*. Baeldung. Retrieved November 28, 2023, from https://www.baeldung.com/cs/kiss-software-design-principle

Baluch, A. (2023, November 2). *How To Change Your IP Address (4 Different Ways In 2023)*. Forbes. Retrieved November 29, 2023, from https://www.forbes.com/advisor/business/software/how-to-change-ip-address/

Basily, A. (2010). *Road Damage*. Dataset Ninja. Retrieved November 29, 2023, from https://datasetninja.com/road-damage

Basily, A. (2020). *Road Damage*. Kaggle. Retrieved November 29, 2023, from https://www.kaggle.com/datasets/alvarobasily/road-damage

Bryant, A. (2012, October 15). *MATLAB, R, and Julia: Languages for data analysis - O'Reilly Radar*. Radar. Retrieved November 29, 2023, from http://radar.oreilly.com/2012/10/matlab-r-julia-languages-for-data-analysis.html

Burns, H. (2016, March 10). *Legal Guidelines For The Use Of Location Data On The Web — Smashing Magazine*. Smashing Magazine. Retrieved November 27, 2023, from https://www.smashingmagazine.com/2016/03/location-data-web-development-and-the-law/

Can the Creators of Smartphone Apps Be Held Liable in Distracted Driving Cases? (n.d.). Leighton Law. Retrieved November 27, 2023, from https://leightonlaw.com/can-the-creators-of-smartphone-apps-be-held-liable-in-distracted-driving-cases/

Carlier, M. (2023, March 14). *Vehicle-miles of travel across all U.S. roads*. Statista. Retrieved November 27, 2023, from https://www.statista.com/statistics/185579/us-vehicle-miles-in-transit-since-1960/

Centers for Disease Control and Prevention. (2022, December 28). *Impaired Driving: Get the Facts | Transportation Safety | Injury Center*. CDC. Retrieved November 28, 2023, from https://www.cdc.gov/transportationsafety/impaired_driving/impaired-drv_factsheet.html

Chatterjee, S. (2020, October 18). *Top C/C++ Machine Learning Libraries For Data Science*. HackerNoon. Retrieved November 29, 2023, from https://hackernoon.com/top-cc-machine-learning-libraries-for-data-science-nl183wo1

Clipground. (2023, June 16). *Windshield clipart*. Windshield clipart. Retrieved November 29, 2023, from https://clipground.com/windshield-clipart.html

Cobb, M. (n.d.). *What is the RSA algorithm? Definition from SearchSecurity*. TechTarget. Retrieved November 29, 2023, from https://www.techtarget.com/searchsecurity/definition/RSA

The Computer Language 23.03 Benchmarks Game. (2023, March). *Python 3 vs C gcc - Which programs are fastest?* Which programming language is fastest? Retrieved November 29, 2023, from https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/python3-gcc.html

Dalsaniya, R. (2021, September 22). *Pothole Detection Dataset*. Pothole Detection Dataset. Retrieved November 29, 2023, from https://www.kaggle.com/datasets/rajdalsaniya/pothole-detection-dataset?select=README.roboflow.txt

Davies, A. (2019, January 30). *CarVi Wants to Help Cure the Common Pothole*. WIRED. Retrieved November 29, 2023, from https://www.wired.com/story/carvi-pothole-detection/

Dianping, M. (2023, January 13). *meituan/YOLOv6: YOLOv6: a single-stage object detection framework dedicated to industrial applications.*

GitHub. Retrieved November 29, 2023, from http://github.com/meituan/yolov6

Edmonds, E. (2022, March 1). *Cost of Pothole Damage*. | AAA Newsroom. Retrieved November 27, 2023, from https://newsroom.aaa.com/2022/03/aaa-potholes-pack-a-punch-as-drivers-pay-26-5-billion-in-related-vehicle-repairs/

Fitzgerald, A. (2022, May 30). *Tailwind CSS: What It Is, Why Use It & Examples*. HubSpot Blog. Retrieved November 29, 2023, from https://blog.hubspot.com/website/what-is-tailwind-css

Flutter Guys. (2023, July). *Flutter Clean Architecture - Learn By A Project | Full Beginner's Tutorial*. YouTube. Retrieved November 28, 2023, from https://www.youtube.com/watch?v=7V_P6dovixg

Flutter_platform_widgets: Flutter Package. (2023, November 17). Flutter_platform_widgets: Flutter Package. https://pub.dev/packages/flutter_platform_widgets

Flutter pros and cons – why use it in 2023? (n.d.). Flutter pros and cons – why use it in 2023? https://www.netguru.com/blog/benefits-of-flutter

Flutter: The first UI platform designed for ambient computing. (2019, December 11). Flutter: The first UI platform designed for ambient computing. https://developers.googleblog.com/2019/12/flutter-ui-ambient-computing.html

Flutter vs. react native: Which one to choose in 2023? (2023, August 18). Flutter vs. react native: Which one to choose in 2023? https://www.simplilearn.com/tutorials/reactjs-tutorial/flutter-vs-react-native

FutureLearn. (n.d.). *What is real-time computing?* FutureLearn. Retrieved November 29, 2023, from

http://www.futurelearn.com/info/courses/embedded-systems/0/steps/64783

FutureLearn. (2022, October 25). *What is real-time computing?* FutureLearn. Retrieved November 27, 2023, from http://www.futurelearn.com/info/courses/embedded-systems/0/steps/64783

Geewax, J. (2021). *API Design Patterns*. Manning.

Geolocation Privacy Legislation - LAW. (2021, March 19). GPS. Retrieved November 27, 2023, from https://www.gps.gov/policy/legislation/gps-act/

Google. (n.d.). *TensorFlow Lite | ML for Mobile and Edge Devices*. TensorFlow. Retrieved November 29, 2023, from http://www.tensorflow.org/lite

Google announces flutter 1.0, the first stable release of its cross-platform Mobile Development Toolkit. (2018, December 5). Google announces flutter 1.0, the first stable release of its cross-platform Mobile Development Toolkit. https://www.androidpolice.com/2018/12/05/google-announces-flutter-1-0-the-first-stable-release-of-its-cross-platform-mobile-development-toolkit/

Google starts a push for cross-platform app development with flutter SDK. (2018, Feb 27). Google starts a push for cross-platform app development with flutter SDK. https://arstechnica.com/gadgets/2018/02/google-starts-a-push-for-cross-platform-app-development-with-flutter-sdk/

Harrison, M. (2014, April 12). *Heatmap of Toronto Traffic Signals using RGoogleMaps*. R-bloggers. Retrieved November 29, 2023, from https://www.r-bloggers.com/2014/04/heatmap-of-toronto-traffic-signals-using-rgooglemaps/

HG.Org. (2023). *Liability for Distracted Driving Accidents and Tech Companies*. HG.org. Retrieved November 28, 2023, from https://www.hg.org/legal-articles/liability-for-distracted-driving-accidents-and-tech-companies-41903

Intel. (2021, October 20). *Intel Launches Intel Unnati Program to Advance Emerging Technology...* Intel. Retrieved November 29, 2023, from https://www.intel.com/content/www/us/en/newsroom/news/intel-launches-unnati-program-india.html#gs.1l4uvg

Intel Unnati Training Program. (2023, August). *Pothole Detection Dataset*. Pothole Detection Dataset. Retrieved November 29, 2023, from https://universe.roboflow.com/intel-unnati-training-program/pothole-detection-bqu6s

Introduction • Docs • SvelteKit. (n.d.). SvelteKit. Retrieved November 29, 2023, from https://kit.svelte.dev/docs/introduction

JuliaHub, Inc. (n.d.). *AI · Julia Packages*. Julia Packages. Retrieved November 29, 2023, from https://www.juliapackages.com/c/ai

Karpinski, S. (2013, February 20). *Re: [julia-users] Suspending Garbage Collection for Performance...good idea or bad idea?* Google Groups. Retrieved November 29, 2023, from https://groups.google.com/g/julia-users/c/6_XvoLBzN60

Khlebovich, P. (n.d.). *IP Webcam - Apps on Google Play*. Google Play. Retrieved April 14, 2024, from https://play.google.com/store/apps/details?id=com.pas.webcam

Kuhlman, D. (2011). *A Python Book: Beginning Python, Advanced Python, and Python Exercises*. Platypus Global Media.

Li, D. (2023, June 16). *\/*. YouTube. Retrieved November 27, 2023, from http://friendlyuser.github.io/posts/tech/rust/computer_vision_in_rust

Liability for Distracted Driving Accidents and Tech Companies. (n.d.). HG.org. Retrieved November 27, 2023, from

https://www.hg.org/legal-articles/liability-for-distracted-driving-accidents-and-tech-companies-41903

The Linux Foundation. (2023, June 16). *PyTorch Mobile End-to-end workflow from Training to Deployment for iOS and Android mobile devices*. PyTorch Mobile. Retrieved November 29, 2023, from http://pytorch.org/mobile/home

Loading data • Docs • SvelteKit. (n.d.). SvelteKit. Retrieved November 29, 2023, from https://kit.svelte.dev/docs/load

Lyu, W. (2023, April 17). *lyuwenyu/RT-DETR: Official RT-DETR (RTDETR paddle pytorch), Real-Time DEtection TRansformer, DETRs Beat YOLOs on Real-time Object Detection. 🔥 🔥 🔥*. GitHub. Retrieved November 29, 2023, from https://github.com/lyuwenyu/RT-DETR

Martin, S. (2022, January 24). *How far should you look ahead while driving?* Jerry. Retrieved November 28, 2023, from https://getjerry.com/questions/how-far-should-you-look-ahead-while-driving

marwaMejri. (2023, June 13). *Flutter Clean Architecture [1]: An Overview & Project Structure*. DEV Community. Retrieved November 28, 2023, from https://dev.to/marwamejri/flutter-clean-architecture-1-an-overview-project-structure-4bhf

Max on Flutter. (2023, June 8). *Flutter Clean Architecture: The Domain Layer*. YouTube. Retrieved November 28, 2023, from https://www.youtube.com/watch?v=AlOVRC6eJqE

Mehta, S. (2021, September 2). *S.O.L.I.D. Principals. Introduction | by Sanjay Mehta*. Medium. Retrieved November 28, 2023, from https://medium.com/@sanjayofficial94/s-o-l-i-d-principals-41a5b3b604eb

Meta AI. (n.d.). *PyTorch Mobile*. PyTorch Mobile. Retrieved November 27, 2023, from https://pytorch.org/mobile/home/

Microsoft. (2014, May 1). *COCO - Common Objects in Context*. COCO - Common Objects in Context. Retrieved November 29, 2023, from http://cocodataset.org/#home

Modulor Inc. (n.d.). *Mojo programming manual*. Modular Docs. Retrieved November 29, 2023, from https://docs.modular.com/mojo/programming-manual.html

National Highway Traffic Safety Administration. (2015, August). *Why your reaction time matters at speed*. SAFETY 1N NUMB3RS. Retrieved November 28, 2023, from file:///home/lumi/Downloads/S1N_Speeding-August2015_812008.pdf

Nienaber, S., Booysen, M.J., & Kroon, R.S. (2015, July). Detecting potholes using simple image processing techniques and real-world footage. *SATC*. 10.13140/RG.2.1.3121.8408

NumPy. (n.d.). *About Us*. NumPy. Retrieved November 29, 2023, from https://numpy.org/about/

NVISION Eye Centers. (2022, November 28). *Driving With Visual Impairments: Statistics & Facts*. NVISION Eye Centers. Retrieved November 28, 2023, from https://www.nvisioncenters.com/education/driving-with-visual-impairments/

OpenMMLab. (2023, June 16). *RTMDet: An Empirical Study of Designing Real-Time Object Detectors*. RTMDet: An Empirical Study of Designing Real-Time Object Detectors. Retrieved November 29, 2023, from http://github.com/open-mmlab/mmyolo/blob/dev/configs/rtmdet/README.md

Pima County Department of Public Works. (n.d.). *Roadway Maintenance Response Times*. Pima County. Retrieved November 28, 2023, from https://www.pima.gov/1000/Roadway-Maintenance-Response-Times

Pothole.info. (2023, June 6). *The Pothole Facts*. Pothole.info. Retrieved November 27, 2023, from https://www.pothole.info/the-facts/

Resource-oriented design | Cloud APIs. (n.d.). Google Cloud. Retrieved November 29, 2023, from https://cloud.google.com/apis/design/resources

Sandler, M., & Howard, A. (2018, April 3). *MobileNetV2: The Next Generation of On-Device Computer Vision Networks*. Wikipedia. Retrieved April 14, 2024, from https://research.google/blog/mobilenetv2-the-next-generation-of-on-device-computer-vision-networks/

Schimansky, T. (2024, January 15). *TomSchimansky/TkinterMapView: A python Tkinter widget to display tile based maps like OpenStreetMap or Google Satellite Images.* GitHub. Retrieved April 14, 2024, from https://github.com/TomSchimansky/TkinterMapView

Shah, J. (2023, May 18). *What's New in Google's Flutter 3.10 and Dart 3: A Dive into the Newest Update*. Everything about flutter 3.10: New features and updates. https://radixweb.com/blog/flutter-3-10-update

Srinivasan, B. (2023, October 2). *Top Object Detection Models in 2023 | Model Selection Guide sponsored by Intel*. YouTube. Retrieved November 29, 2023, from http://www.youtube.com/watch?v=dL9B9VUHkgQ

StackOverflow. (2022). *Stack Overflow Developer Survey 2022*. Stack Overflow Insights. Retrieved November 29, 2023, from https://survey.stackoverflow.co/2022/

Stroustrup, B. (1997). *The C++ programming language*. Addison-Wesley.

Submissions | 2022 IEEE International Conference on Big Data. (n.d.). Crowdsensing-based Road Damage Detection Challenge (CRDDC2022). Retrieved November 29, 2023, from https://crddc2022.sekilab.global/submissions/

Tan, M., Pang, R., & Le, Q. V. (2019, November 20). *EfficientDet: Scalable and Efficient Object Detection*. arXiv. Retrieved April 14, 2024, from https://arxiv.org/abs/1911.09070

TensorFlow Lite | ML for Mobile and Edge Devices. (n.d.). TensorFlow. Retrieved November 27, 2023, from http://www.tensorflow.org/lite

Ultralytics. (2022, November 22). *ultralytics/yolov5: YOLOv5 🚀 in PyTorch > ONNX > CoreML > TFLite*. GitHub. Retrieved November 29, 2023, from https://github.com/ultralytics/yolov5

Ultralytics. (2023, January). *ultralytics/ultralytics: NEW - YOLOv8 🚀 in PyTorch > ONNX > OpenVINO > CoreML > TFLite*. GitHub. Retrieved November 29, 2023, from http://github.com/ultralytics/ultralytics

United States Census Bureau. (2023, June 29). *2020 Census Urban Areas Facts*. Census Bureau. Retrieved November 28, 2023, from https://www.census.gov/programs-surveys/geography/guidance/geo-areas/urban-rural/2020-ua-facts.html

United States Department of Transportation. (2017, May 31). *National Household Travel Survey Daily Travel Quick Facts*. Bureau of Transportation Statistics. Retrieved November 27, 2023, from https://www.bts.gov/statistical-products/surveys/national-household-travel-survey-daily-travel-quick-facts

United States Department of Transportation. (2020). *Road Condition*. Bureau of Transportation Statistics. Retrieved November 28, 2023, from https://www.bts.gov/road-condition

Vecteezy. (n.d.). *Happy family car for banner. Car sedan for travel. Asphalt road near the green grass and mountain under clear sky for winter*

travel. Copy Space Flat Vector. Vecteezy. Retrieved November 29, 2023, from https://www.vecteezy.com/vector-art/16188007-happy-family-car-for-banner-car-sedan-for-travel-asphalt-road-near-the-green-grass-and-mountain-under-clear-sky-for-winter-travel-copy-space-flat-vector

VXL. (n.d.). *VXL - C++ Libraries for Computer Vision*. VXL - C++ Libraries for Computer Vision. Retrieved November 27, 2023, from http://vxl.github.io

What is Svelte? (n.d.). Educative.io. Retrieved November 29, 2023, from https://www.educative.io/answers/what-is-svelte

Yiu, W. K. (2022, July 6). *WongKinYiu/yolov7: Implementation of paper - YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. GitHub. Retrieved November 29, 2023, from http://github.com/WongKinYiu/yolov7

Appendix

Permissions

Permission granted from Horizon Vectors by Vecteezy.com for using related vector art in this document given appropriate accreditation.

Clipground grants permission for using their clipart in personal, non-commercial uses.

Links

All the repositories for the project can be found at https://github.com/AI-Pothole-Detection