**Distributed Android Seeing Eye Application with Obfuscation**

David, A, Spellman

SCI, University of Pittsburgh, das320@pitt.edu

Fan, Shen

SCI, University of Pittsburgh, das320@pitt.edu

Navigation of public places is a central issue that blind and visually impaired people are faced with. This can prove especially challenging when it is a public place that the individual is unfamiliar with. A cane or guide dog can provide a certain amount of assistance that makes navigating unfamiliar public settings possible, but a cane and guide dog can provide limited information in certain situations. Using a smart device as a guide using object recognition and depth perception provided through a combination of machine learning and knowledge base approaches.

# **1 BACKGROUND**

Successful seeing eye applications have been produced in the past; however, the most successful of these approaches required the user to carry specialized sensors on them, and utilized a single privately owned server to perform machine learning. Other approaches limit the machine learning to a single smart device to perform offline object detection; however, these localized models are severely limited in power compared to models run on a server. The other issue with running all of the machine learning on the smart phone is power consumption, and the fact that the machine learning aspect cannot be supplemented with an expert system to provide more detailed feed back to the user. However, an offline approach does have merit when it comes to use in an environment where network access is not readily available. In the case of a public setting such as a town it would be more ideal to take advantage of higher bandwidth network connections to take a distributed approach since you can achieve higher recognition accuracy and add additional advanced image processing such as depth perception. In a town setting with a good enough network connection such as reliable 5G a seeing eye application might be able to achieve higher frame rates while providing improved feedback from an expert system with a built up knowledge base of landmarks accumulated from client data. However, In a distributed or local setting privacy can be an issue. Privacy issues could come in two forms, other applications on the smart device snooping on image data from the seeing eye application, and the cloud provider collecting any sensitive data that is sent to the cloud.

# **2 DESIGN/EXPERIMENTAL SETUP**

The purpose of our design is to offer a seeing eye application that is accurate, responsive, keeps private information from leaking through image data, does not require more equipment than a single smart device, and does not require extra equipment to be installed in public places where the technology is to be used. The purpose of this is to make a solution that is scalable for many users, will not be difficult to use, and be quickly scaled for use in towns or cities.

Our proof of concept for a client-server architecture seeing eye application consists of three major components, the android application, the object detection models, and the server. The application uses the CameraX API to take a constant stream of images, one about every five milliseconds. The application has buttons to start and stop the recognition process along with a spinner control to switch between the distributed and local application modes. The local application mode performs object recognition directly on the phone using tensorflow lite, and displays the classifications to the screen. The distributed mode does not use the local object recognition model, but instead loads a facial recognition tensorflow lite model to perform facial recognition on the phone with. The facial recognition model outputs a feature map that contains scores for each region of the image that is potentially a face, and anything above a threshold value of 0.5 is extracted as a possible face. This facial recognition model will classify up to 896 faces in any images and will produce a list of feature points for each face. Using these eight points a bounding box can then be calculated by finding the minimum and maximum x coordinates for each face detected along with the minimum and maximum values for the y coordinates. Obfuscation of the image is then performed on the bit map object by setting all pixels between the minimum and maximum values for both x and y to 0, or blacking out all of those pixels.

Once obfuscation is performed on all the regions classified as faces, the distributed mode uses xmlrpc to send the image to the server. To send the image to the server the bytes encoding the image are turned into a base sixty-four encoding to help keep the size of the data packet small. When the image reaches the server the base sixty-four string is converted to a numpy array and object recognition is run with a more powerful model using pytorch. The server sends back a response containing the label for the object detected, otherwise it will return a string indicating that no recognizable objects were detected. For the purpose of logging in order to perform the tests, the current system time is collected on the client side in milliseconds when an image is captured, and when the result of the object recognition has returned. The difference can then be taken between these two times in milliseconds to get the amount of overlapped time. These times are then each written out to a csv file along with the result of the classification. A separate csv is written out for each of the two modes for producing graphs to compare the performance.

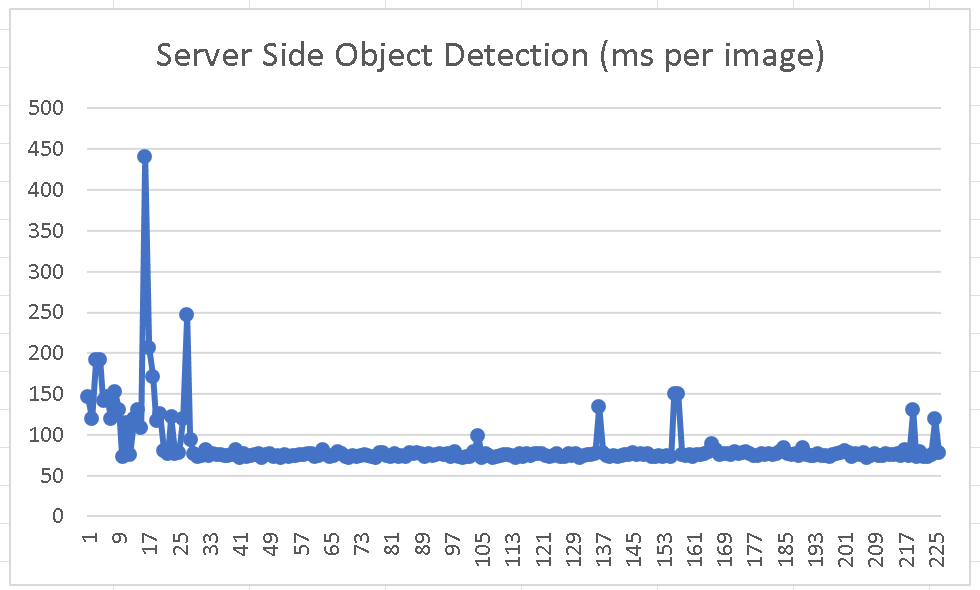
In a second version of the application Flowfence is additionally integrated for providing greater privacy on the client side. Flowfence provides an API through which the functions in the application that deal with the image data can be run in a sandbox. This is done by placing all privacy concerned functions into a quarantine module. The quarantine module contains methods where the operations are performed on the private data, and the quarantine methods are invoked using the flowfence context which provides opaque data handles. The purpose of adding the quarantine module and the sandbox is to make it harder for other applications on the client side to snoop on the data coming from the camera.

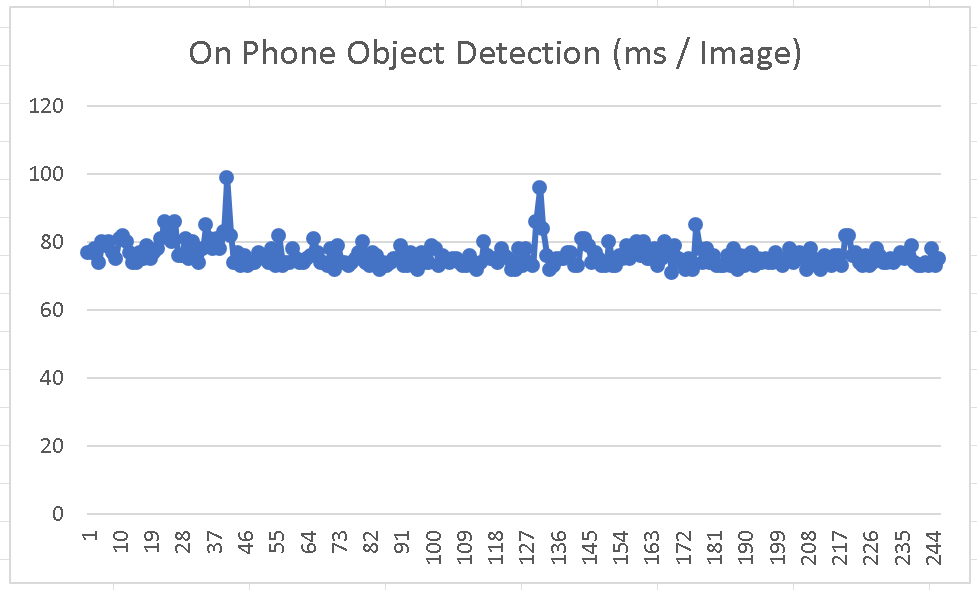
# **3 CONTRIBUTIONS**

David contributed to the project by initially researching what has recently been accomplished when it comes to using machine learning and artificial intelligence for solving the seeing eye problem with computing. David also worked on the android client-side code, integrating the models into the application, integrating a version of the application with Flowfence quarantine modules, and running the tests. Fan worked on the server side xmlrpc code and on precuring object detection machine learning models.

# **4 RESULTS AND CONCLUSIONS**

Our proof of concept changed from specifically focusing upon the challenging issue of making a seeing eye application that can help navigate stairs to more generic object recognition do to a lack of good public available datasets to work from, and the fact that integrating the existing support vector machine model would have proved very time intensive. Getting the CameraX API to work, and integrating the object recognition and facial recognition models proved more difficult than expected. Even establishing the xmlrpc communication successfully between the android client code and the python server code took a deal of trouble shooting before the responses would return successfully. No additional sensor readings were used on top of the camera do to time restraints and issues running the tests. The flowfence version was also not used for testing do to issues getting it to run with the available version of flowfence along with the CameraX API code. Since a commercialized version of Flowfence was never produced and the currently available version of Flowfence was for research purposes it is no longer compatible with newer Android modules such as CameraX. In order to do a working integration, the Flowfence code would need to be updated to be compatible with the current Android version.





The conclusion that can be made from this proof of concept is that there is certainly a potential for a distributed mode for a seeing eye application that uses image data. The application is not as responsive in distributed mode as when it runs in local detection mode, but the latency is low enough in distributed mode to make it feasible to allow for at least 100 milliseconds worth of computation time on the cloud and still have the application be more than responsive enough for real time use. The facial obfuscation does not prevent other objects from being recognized in the images, and does not significantly effect the performance.

Future improvements that could be made to this proof of concept would be to add lossless compression to the image data in order to further reduce the number of bytes that must be sent over a network. Models or algorithms could be added to the server side to support depth perception for gaging how far the camera is from classified objects. To further the possibility for depth perception without a device supporting special camera features, the application could prompt the user to tilt the camera about by rotating their arm or wrist to provide a stream of images from slightly different angles. A knowledge base approach could be added to the server side in order to store feature maps of landmarks for commonly visited public locations to help the user obtain more specific information about their surroundings. Another very important aspect that a polished version would want to include would be support for auditorial feedback as well as voice recognition. A polished application would also want to ensure convenient use with a mobile screen reader. If an expert system was used on the server side for distributed mode then the expert system could interact with the user and ask questions in order to specifically tailor feedback to where they are trying to get to, or to provide any specific visual feedback that will be currently useful to the user.

A picture containing text, cat, floor, indoor

Description automatically generated