**Methodology**

**Problem Statement**

Human sight and the ability to instantly recognise what they are looking at, what objects are visible or how many objects are in view are all tasks that occur daily and frequently. It’s not until the repetitiveness of this nature becomes noticeable the realisation of how mundane doing these tasks are and the need for a more straightforward process is required. Furthermore, human sight is perspective based and subjective to what a person thinks they saw. A constant error rate will always be prevalent due to personal reasoning. What if these issues could be addressed?

These tasks mentioned and many others are frequent throughout industries, and the need for an effective solution that adapts to various use cases is high. Computer vision aims to address these concerns by automating human sight into a machine-processable item that can solve issues like image classification, object detection, and object counting. Numerous tools, models and frameworks exist in the environment of ML that seek to make implementing a computer vision problem workflow for different use cases achievable.

FAIMS are an example of an industry need for ML workflow to address computer vision problems and the critical reasoning behind the purpose of this thesis. Since their application aims to simplify the process of collecting fieldwork data in a virtual notebook, the extension into ML will further benefit this goal when handling image samples. The final important aspect of their use case is the integration into their application which was designed for various systems like AOS, IOS and web applications. A core consideration of this problem is implementing ML onto devices with limited hardware. Specifically, two areas of computer vision are the desired focus for implementation:

Whilst, these are both different problems with different requirements, as seen in research, they are both computer vision problems, with object counting being a sub-problem of image classification. The goal of the proposed workflow would be to address both of these problems. The intent would be to find the closest related workflow for both of these issues so that approaches, ideas, tools and frameworks can be shared to create the most seamless possible workflow and improve the possibility of implementing both features.

So based on all this information, we can formulate the research question:

How will a seamless ML workflow be implemented onto devices with limited hardware?

**Sub-questions**

It is clear the question is vague and involves many core components to answer it correctly. Proceeding with this question and tackling it effectively, it will be deconstructed into sub-questions that correspond to core issues. Answering these sub-questions will answer the overarching research question. A high-level look into the approaches to testing and answering these sub-questions will be provided, from which the /ref will give a more in-depth analysis of the whole system’s design.

**Reflecting on the Literature Review**

The current on-device ML environment survey conducted in the literature review revealed a lot about the state of ML for image classification and object counting problems and the overall intricacies of an on-device ML workflow. The discoveries and assumptions can help future research by providing a concise analysis of the environment, allowing researchers to improve upon it or adapt it to their study. The key takeaways from the literature review are:

\begin{itemize}

\item The steps and processes involved in an image classification problem

\item The steps and processes involved in an object counting problem

\item Understanding the reasoning of CNN and CNN based network’s preferential use in ML models for computer vision problems

\item The architectures used for on-device ML – Examining robust designs and determining which best suits FAIMS use case

\item An analysis of the frameworks/tools used for on-device ML – TensorFlow provides strong tools which effectively address on-device ML

\item Performance analysis of on-device ML frameworks/tools

\item Analysis of ML models for image classification problem – Evaluating the performance of a selection of ML models for image classification

\item Analysis of ML models for object counting problem – Evaluating the performance of a selection of ML models for object counting

\item Insight into the data requirements for on-device ML

\item Understanding the importance and robustness in transfer learning a pre-trained ML model

\end{itemize}

Based on the teachings and evaluations made during the literature review, questions for core aspects of on-device ML can be proposed. Additionally, the approaches to the questions will be presented to provide a clear plan and workflow for implementing on-device ML in the FAIMS application.

**ML Model**

*Will the model be made from the ground up, or will it be selected from a pre-trained Model?*

Based on the research, it seems more beneficial to the project’s goals to use a pre-trained model rather than a purposely created one. Considering the sheer requirements for creating a model, this would only slow down this research project’s true intentions of on-device ML. Additionally, since addressing both image classification and object detection problems would require separate models tailored to handle each situation, this again would impact the project’s time constraint. However, creating a model would provide complete transparency over the functionality; this is only one payoff where ultimately, the model could be inaccurate, and thus a waste of time \cite{hussain\_study\_2019}. The transfer learning process seems highly effective in retraining a model for the desired purpose, with results being more substantial than a model created \cite{hussain\_study\_2019}. These discussed models were designed for accurate classifications and detection of objects on extensive datasets. Adapting those models for the FAIMS fieldwork objectives would present an easier workflow and better accuracy results.

*Which model will be chosen?*

These additional queries need to be considered before selecting a model.

Performance of the model?

Size of the model?

Hardware requirements of the model?

Is the model obscure or well-known?

Fortunately, the models already presented in the literature review are relatively well-known and researched, meaning there are plenty of resources and support for implementing these tools. However, determining the best model for the project will involve analysing and testing the models. Reviewing the number of parameters each model needs to train will provide a good indication of the scale. Additionally, based on the time and complexity of the project, ideally, $2\sim4$ models for each problem will be set up in a test environment where it will be tested against a nominated dataset. The results provided in the form of graphs and tables will be analysed to compare accuracy, error rate, speed, and device performance after the testing. Based on the outcome data, an informed choice will be made. Also, a sub-criteria for the nominated choice will be how seamlessly the image classification and object counting models correlate.

*Which model best suits on-device ML?*

Determining this will involve selecting models for each problem being tested on a dedicated workstation with better hardware than a mobile device, which will funnel out insufficient models. Furthermore, depending on the application architecture, there could be two different sets of steps to handling testing. In architectures that involve a server, the inference is on the device with updates from the server, or it includes the latency of requests and responses made to the server if it’s purely server-side. Regardless, a commonality is an interaction with the server where latency issues can be amplified if the ML model is too large or overly complex in its tasks. Moreover, for the client-server models and the increased time of downloading a large model, since it conducts inference on the device, an inefficient model will also extend the use time of the system, ultimately impacting usability. The pure client-side approach also needs to be concerned about ML models that are intensive on the user’s hardware as it will slow the performance of the device. After initial ML model testing in a test environment, the selected models will be loaded onto devices to see the performance there.

*How will an object detection model be adapted to the object counting problem?*

Object counting is essentially another process of the object detection problem. Once the object has been segmented or classified into a bounding box, the number of boxes or segmentations associated with a defined class will be counted. In addition, the criteria discussed in the literature review of whether either APIs can be utilised \cite{ozlu\_tensorflow\_2022,noauthor\_welcome\_2022}. These APIs will benefit in developing a workflow for object counting.

*How accurate will the classifications and counting of objects be?*

Currently, there is no clear indication of how the models will perform the required tasks of FAIMS. Only through testing the models on validation data will comparable results be generated. However, the literature review provides insight into the models’ performance on their originally trained datasets. Each of the models did achieve great accuracy results, which would certainly satisfy the needs of this project. A desirable accuracy for this project would be $\geq50\%$ as the pressing concern is implementing a complete on-device workflow.

**System Architecture**

*Which architecture is best for providing a robust application of on-device ML?*

Though each on-device ML system design discussed in the literature review does present an argument for its adoption, some considerations need querying before committing to a structure. It seems that architectures that consist of servers for hosting the ML model are the most researched and tested designs. Pure on-device ML is an experimental paradigm that appears to parallel the increase in hardware capabilities of edge devices. Though hardware performance has increased and thus the interest in this design, the issue of the prevalence of older devices that don’t satisfy the performance requirements prevent this approach from becoming currently mainstream and an ideal implementation choice. A pure server-side design, whilst addressing this concern by not relying on the hardware of an edge device but utilising the server for all its computations, does restrict the use cases for the application. A complete dependency on the server would limit any offline capabilities achievable, as a constant connection to the server is needed. This reliance would counteract the ability of the FAIMS application as remote fieldwork data collecting would likely not be feasible due to the lack of a connection.

Furthermore, the reliance on a service for hosting the model means the application depends on continuous uptime for it to be functional. If any uncontrollable issues were to arise, resulting in the service’s servers being down, the application would be down as well. Hence, the client-server design is the most applicable for FAIMS. It balances both pure client and server-side approaches by having a middle ground where the model is loaded onto the device for inference only. Meaning no computational power is needed for the intense training of the model, and it only interacts with the server when new unseen data that the model can’t classify or count needs to be sent to the server to retrain it. Subsequently, the updated model will be reloaded onto the device through an update from the server. The application can be used offline, in remote settings and when servers are down. This design correlates to the objectives of the FAIMS applications whilst also providing a robust, desired experience.

*Which architecture will provide offline on-device ML?*

Research indicates that both the client-server and client-side designs achieve offline ML capabilities. The server-side architecture requires a constant connection to the server as requests to the model to make classifications and count objects are made, and responses are sent back. This vital connection means offline ML with this design is not possible.

*What framework/tools will be used to achieve on-device ML?*

The criteria for determining the frameworks/tools to be used for implementation will be how well the tools interface with each other, do they benefit the goal of on-device ML and do they help address the target problems of image classification and object counting. TensorFlow seems to be a strong option, as research highlights. The platform has robust APIs, allowing it to access models for image classification and object detection/counting problems. Additionally, it caters to multiple platforms by providing tools that compress the TensorFlow models, making them more compatible with their intended devices. TensorFlow Lite compresses the models for mobile devices, and TensorFlowJS compresses/ makes ML available for JS systems like web applications. These uses accommodate precisely the needs of the FAIMS application AOS, IOS and web applications. Finally, TensorFlow integrates effectively with backend server services like ML Kit and Google Cloud due to its ties with Google. Though TensorFlow does provide a strong argument for its use, other platforms can be tested to determine any synergies that outperform TensorFlow and benefit the creation of an on-device workflow.

**Data Requirements**

*How much data is required for the task?*

The amount of data needed to invoke the transfer learning process sufficiently to achieve ideal accuracy rates is undetermined without testing. The testing process of the training data would adhere to a similar process presented in this study \cite{cho\_how\_2016}. In this research, they had a dataset with 6000 images, which they incremented the training data size to determine the optimal amount of data needed to retrain the model. Determining the total amount of training data from the dataset would require invoking the validation split. The validation split, where the dataset is split between the training data and the validation data, is necessary as the training data is integral to the model learning how to complete its task on unseen data. Thus, the training data needs to be significantly more extensive than the validation data. Once the training data size is resolved, the increments $\mathnormal{i}$ will start at a specified value and double until the total length of training data $\mathnormal{N}$.

The model will be trained and tested in the test environment with each increment to see the accuracy results. These results will determine the training data size. As for the size of the initial dataset, this can vary. Immediately, since the transfer learning technique does not require much data from the dataset as the model has already been pre-trained on a larger dataset, that narrows down options to include smaller datasets that target the specific FAIMS use case better. Larger datasets like ImageNet or COCO don’t have to be exclusively used or sourced.

*Which dataset will be used?*

The datasets specified in the literature review can be used as they are somewhat relevant to the use case of FAIMS. Ideally, datasets will be provided by the stakeholder CSIRO, which will be directly relevant to the target scenario. Alternatively, data collecting can also be pursued. Creating a robust classification system that captures the essence of the items, such as variations, types, and other aspects, will generate valuable, relevant data. This process will involve taking images, annotating them with labels correlating to the classification system, and storing them in a database.

*Do any data enhancing techniques need to be employed?*

Image pre-processing will be utilised on the initial data to provide some variety and to ensure any discrepancies in the image data are addressed. Whilst, this process may not be necessary, it will ensure a more robust and accurate image classification and object detection system. Data augmentation techniques will also not be noticeable until the model is tested on the training data. These techniques will be used if signs of overfitting or underfitting are prevalent. The image data will undergo modifications like flipping the image horizontally or vertically, cropping the image, image rotation and many others to address the issue. Epochs and the batch size will be initially set values regardless. However, since a vast number of epochs could be a cause of overfitting, a trial-and-error method will be used to determine the optimal amount of epochs with consideration to the performance impacts of an overly substantial quantity.

**Project Plan**

Based on the methodology and preliminary design, a plan for the project will be drafted to present the reader with a timeline of events and expectations to be completed for the whole \textbf{Thesis}.

Ultimately, the final goal of this project will be to deliver FAIMS with a complete workflow and implementation for on-device ML for image classification and object counting problems.

**Vertical Sprints**

Throughout the \teftbf{Thesis} vertical sprints have been applied to take a slice of the project and thoroughly research it before moving on to the next issue. This same principle will be used for the rest of the \textbf{Thesis} with the deliverables of the project. Utilising this project methodology, the current project deliverables that have been worked on are:

\begin{itemize}

\item Thesis draft

\item Project presentation

\item Log that provides a rough overview of the current project journey

\item Demo implementation of image classification on a web application

\item Demo implementation of training the Sequential ML model for a specified task in a test environment and preparing it for on-device ML

\end{itemize}

Deliverables that need to be addressed in the remainder of the project through vertical sprints are:

\begin{itemize}

\item Nominating an ML model through testing

\item Initial model training through transfer learning

\item On-device ML for image classification problem

\item On-device ML for object counting problem

\item A backend server that is responsible for hosting the ML model

\item The server’s ability to retrain the model with new data

\item Pushing model updates to user devices

\item Integration of user experience into either test front-end or FAIMS application

\end{itemize}

The elicited requirements in the appendix emphasise the system’s expectations and outline the deliverables listed. The functions of the system are highlighted in table ##. These functions are measured under performance metrics which are presented in table ##. Performance requirements that cannot be measured are then described in table ##.

**Project Timeline**

The project constraint of time needs to be addressed to provide these deliverables fashionably. The primary project developments will occur in Semester 1 of 2023, consisting of 13 weeks. The timeframe will be managed into a weekly plan to ensure consistent advancements happen over this period. Whilst, developments into components of the project have already begun, these are preliminary and are subject to change if testing or progress presents a stronger argument for variations. The weekly plan for this project is as follows:

\begin{itemize}

\item Week 1 – Initial project setup and arrangements

\item Week 2&3 – Vertical Sprint 1: Training, Testing and nomination of ML model for on-device ML

\item Week 4&5 – Vertical Sprint 2: Data collection, Transfer Learning of ML model on dataset

\item Week 6&7 – Vertical Sprint 3: Building the on-device user experience for image classification and object counting

\item Week 8&9 – Vertical Sprint 4: Setting up the backend server to host model, retrain model, push model updates to the user

\item Week 10 – Integration into FAIMS application

\item Week 11 – System testing, bug fixing, refactoring

\item Week 12&13 – Updating \textbf{Thesis} with results and discussion of results

\end{itemize}

Figure GC visualises this plan, highlighting the flow of completed tasks. The mid-semester break, whilst not included in this weekly plan, will be a safety bracket that can be used as a period to provide additional time to aspects of the project plan that have exceeded their quota. This time will also be when revisions to existing completed tasks can be made. These include:

\begin{itemize}

\item Incorporating literature review feedback

\item Updating methodology for any changes

\item Updating preliminary design for any changes to the final design

\end{itemize}

This safety bracket is purely a precaution to ensure anything that is not going to plan can be addressed and not ruin the project’s progress. It is also in anticipation of each week’s commitments to the \textbf{Thesis} project as well as other university needs and personal obligations. Fortuitously, the problems intended to be solved in this project are pretty researched; thus, there is lots of supporting documentation and resources to help with the development if needed \cite{ ozlu\_tensorflow\_2022,noauthor\_welcome\_2022, team\_keras\_nodate-3, team\_keras\_nodate-4 team\_keras\_nodate}.

**Project Development**

Project developments at this stage have contributed to the preliminary design and next phase of this project. The necessary dependencies like Android studio, Android virtual devices, React framework, node.js, npm, Python and Jupyter notebooks have all been set up and prepared for future developments that will be integral to this project. Testing the implementation of the Sequential ML model in a test environment on the Tf flowers dataset from start to finish has been done. Conversion of the model to a mobile device and web application targeted version proceeded (TensorFlow Lite and TensorFlow.js) \cite{ noauthor\_image\_nodate}. A web application demo of the image classification problem with the MobileNetV3Large model and retrained Sequential model using the React framework and JS was developed \cite{sheng\_image\_2021}. Preliminary results were also generated for handling overfitting using data augmentation techniques. A closer fit between the training and validation data was fostered, as seen in figure \textbf{4.3}. Additionally, desirable accuracy results on the two sets were shown, 78.54% for training and 74.93% for validation. These preliminary designs and outcomes will assist in approaching the next phase of this project.

**Requirements**

**Functional Requirements**

The functional requirements represent the expectations of the system’s ability to complete agreed-upon tasks.

**Preliminary Design**

A preliminary design is presented after the methodology’s high-level overview of how this project will be approached, and the research question answered. This design aims to provide a more in-depth, direct attempt at conceptualising a system and answering the research question.

**Nominated ML Model**

Since both problems of image classification and object counting are being addressed, a significant selection factor is a relationship between both models. The more vital it is, the more beneficial the choice will be to a seamless workflow where ideas and approaches can be shared between implementations. Additionally, models designed with resource efficiency concerns are favourable due to edge devices’ limited hardware.

Thus, the chosen model for the image classification problem is \textbf{MobileNetV3} and for the object detection problem \textbf{SSD MobileNetV2}. These models, even though they have different deep learning networks, with MobileNetV3 being a CNN and SSD MobileNetV2 being an SSD, they are still a part of the MobileNet family. Implementation of transfer learning on these models will still follow a similar set of steps common to MobileNet models. Both of these models were built to target low-resource devices like mobile devices, which is noticeable with the small number of parameters needed and the modest size of the models. Server updates with new versions of these models will be much quicker than larger ones. MobileNetV3 also has sub-versions providing flexibility for targeting higher or lower resource-available devices. Though this feature is not present in SSD MobileNetV2, it is still very efficient and optimal for on-device ML. Furthermore, it is supported by integral object detection and objecting counting TensorFlow APIs.

**Dataset**

Ideally, the datasets used in this project will hopefully be provided by the project stakeholder, the CSIRO. If this cannot be accommodated, then a combination of personal data relevant to the use case of the FAIMS application will be collected and the use of online datasets for testing like IBean and Tf flower. Collecting data will involve a classification system like the type of seeds, how many objects of that class exist and the appropriate labelling for each image.

**Transfer Learning of ML Model**

The concept of transfer learning involves taking a pre-trained model on an extensive dataset and retraining it on a new set of relevant data, in this case, data pertinent to the FAIMS use case.

**Image Pre-Processing**

The process of Image Pre-Processing is essentially an early step where the data used for training and validating the model will go through a process of cleaning any noise or unwanted discrepancies. Whilst, this may be a bit redundant for online repositories or professional datasets, which should have robust data, this is preventive and will expand upon the data. This step is mainly to target any collected data issues, such as:

\begin{itemize}

\item Poor image quality samples

\item Poor lighting in image samples

\item Mislabeled data

\item Incorrect data

\item Insufficient data

\end{itemize}

The following pre-processing techniques will be employed to account for these complications:

\begin{itemize}

\item Image correction – fix mislabeled data, fix incorrect image samples

\item Image enhancement – upscale images to improve quality

\item Image restoration – fixing lighting issues, removing image noise

\item Image compression

\end{itemize}

**Feature Extraction**

Feature extraction involves specifying the unique features of an image, which will be helpful to the ML models classifying images and counting objects. This factor is not a significant priority as the model’s network will already handle it. The development of deep learning networks has seen this step become automated and incorporated into the layers of the network.

**Epochs**

As discussed in the methodology, the number of epochs will go through a trial-and-error process to determine the optimal number to prevent overfitting. It has been noted that studies suggest more iterations over the training data will substantially impact the accuracy of the model. The process of epochs looks like this:

**Input:** A infinite amount of Epochs can be set E = {E0, E1, E2, …, E∞}

A finite number of batches can be set B that is less than or equal to the total dataset size N ={B}

**Data Augmentation**

Data augmentation will be crucial if the results indicate a noticeable problem with overfitting or underfitting. If no more data can be provided to the model to ratify the issue, then modifying the image samples will give the model additional training data. The model will perform better on unseen data, which significantly varies from the training set. Preliminary results from testing the Sequential ML model indicated that the model was struggling to garner the same accuracy results on the training and validation data; thus, the loss rate saw a massive distinctive increase, as seen in figure \textbf{4.1}. The problem could have been a cause of setting the epochs too high; however, initial resolution testing saw the use of augmentation techniques like horizontally flipping the image, rotating the image on its axis and zooming into the picture. As seen in figure \textbf{4.2}, the modifications generate much more data for training the model. The model was retrained on the new augmented data, where results show a desirable closer fit of both accuracy and loss rates for the training and validation data, as shown in figure \textbf{4.3}. These results ultimately support using data augmentation techniques to address overfitting and underfitting.

**Training the Model**

All these steps will be applied in the transfer learning process of \textbf{MobileNetV3} and \textbf{SSD MobileNetV2} for the system developed for FAIMS. Based on preliminary testing on the Sequential model, using the Tf flowers dataset and following a similar methodology, for 15 epochs, accuracy results on the training set were 78.54%, and on the validation set were 74.93%. These results are well above this project’s ideal target of 50%, indicating solid results for implementing the system following this methodology.

**System Design**

As seen in table \textbf{A.1} and \textbf{A.2.1}, the initial project requirements were the implementation of on-device ML that followed a well-researched, tested approach and performing on-device ML offline. After conducting a literature review, it seems that the \textbf{client-server architecture} is the ideal choice that satisfies these requirements for the project.

**Frameworks/Tools**

Since the current preliminary system work has been completed using TensorFlow, proceeding with the project will likely be the continued use of TensorFlow. This tool will benefit this project’s on-device ML workflow. It offers a vital synergy with many services like Google cloud products, renowned ML models, and options for various applications of ML. Moreover, as highlighted in the methodology, a shift would be viable during the next phase of this project if a more robust relationship between frameworks and tools is discovered that even further benefits this project’s goals.

**Cloud**

The cloud or server-side for this project will provide hosting for the ML model. Not only will it hold the model, but the server will communicate with the client device to receive unseen data from the client to retrain the model, ensuring that it will constantly satisfy the use case needs of the FAIMS application. The only communication between the client and the server will be when the model incorrectly classifies the image or incorrectly counts the number of objects. The incorrect data, once labelled correctly by the user, will be sent to the server for retraining of the model. Once the model is retrained, the server will send a new, updated version to the client to be loaded onto their device. The research conducted in the literature review suggests Google services as the preferable option for hosting the model due to the support of the TensorFlow framework. Specifically, Firebase ML Kit, a Google service, provides plenty of options and support for deploying ML models and setting up a client-server on-device ML system.

**Client**

The client-side of this system will be responsible for loading the ML model onto their device and conducting inference. Since the model will be loaded onto the device, the system can function offline. When the client has a connection, updates that need to be sent to the server or received from the server can begin. This feature supports the objectives of the FAIMS application being able to collect data in remote environments that possibly have no internet connection.

**System Implementation**

The FAIMS application currently targets three domains: AOS, IOS and Web applications. This on-device ML project will also target these same systems so that integration with the FAIMS application will be seamless.

**AOS Application**

Android users will be provided with a proper native application. Ideally, the experience which will be incorporated into the FAIMS application, after being refined in a test application, will be the user normally proceeding with the steps of filling in a notebook. Now, users add an image sample to their notebook, the system will classify and count the number of objects in the picture rather than having to notate the sample. If an incorrect classification or counting of objects occurs, the user can modify the result to rectify it with the correct answer. Depending on the user’s connection status, when they are connected to the internet, the image with an incorrect result that the user fixed will be sent to the server-side model to retrain it.

**IOS Application**

IOS users will be provided with a proper native application. The experience, besides any variating native features, will be identical to the systematic steps highlighted in the Android application.

**Web Application**

The web application version of this project will provide a unanimous, shared experience for any user accessing the system through the web. The core feel of the system will be reflective of both the AOS and IOS application steps described. This system version needs to be tested to determine whether it can support the client-server architecture or needs to be adapted to a server-side design. Since functionalities of a web application can be loaded to maintain an offline mode, the extent of this needs to be determined. If not all core functionalities relevant to this project can be loaded, then this system version will have to utilise a server-side design. In this scenario, offline capabilities will be limited to AOS and IOS applications.

**Work Environment**

This project will primarily be developed on a Surface Pro 8 laptop with Windows 11, 16 GB of RAM, 1 TB of Solid State Drive storage and an 11th gen Intel Core i7-1185G7 @3.00GHz CPU. This hardware should be more than sufficient to support project development; however, a desktop workstation can help by providing extra computational resources if needed. For Android on-device ML testing, a combination of using Android virtual devices, specifically a Pixel 4, an older Google API 28 device, and a physical Samsung 10+. The Android application will be built in Java using Android Studio. IOS on-device ML testing may prove difficult due to a lack of IOS and MacOS devices that allow for physical device testing and creating IOS virtual devices. Apple devices will need to be sourced to test this system variation or determine whether a MacOS virtual machine can be set up for development. The IOS application will be built in Objective C and Swift. The web application, fortunately, has no hardware dependencies but requires Node.js and node package manager. It will be designed in the React framework using JS.

**Test Environment**

The test environment has been referred to throughout this paper as a place where ML models can be tested. This environment is a Jupyter notebook where the TensorFlow models can be trained, tested and evaluated using the Python 3.9.12 programming language. Once the models have been adequately tested, Python functions can be set up to export the TensorFlow models to TensorFlow Lite and TensorFlow.js models. The ability of Jupyter notebooks to easily display statistical information about the models and Python’s extensive support of various ML libraries is why these tools were chosen.

**Conclusion**

In conclusion, the current state of this research paper introduces the problem of implementing an on-device ML workflow for the FAIMS application. Precisely, the workflow needed to target the issues of image classification and object counting when a user uploaded image samples into their notebook. Effectively approaching this implementation required surveying and reviewing the existing on-device ML environment to understand the choices, options and lessons presented in handling these problems. This insight would further help craft an approach to testing and designing a solution relevant to the FAIMS use case. Conclusively, based on the research findings, a preliminary design has been presented which employs the client and server architecture and utilises the ML models \textbf{MobileNetV3} and \textbf{SSD MobileNetV2} for their respective problems. Additionally, preliminary results have also reaffirmed studies suggesting the effectiveness of data augmentation to help remedy overfitting and underfitting.