Framework Development For Energy-Efficient Mobile Augmented Reality Applications & An Analysis of AI/AR Task Inference Resource Utilization

Research Report

By Syed Safwaan

Advisors
Dr. Marco Brocanelli
Niloofar Didar

CSC 4999 — Departmental Honors Thesis

WAYNE STATE
College of Education

College of Engineering
Wayne State University
Detroit, Michigan, USA
August 15, 2022
Contents

1 Abstract 3

2 Introduction 3

3 Application Development Overview 5
   3.1 ImageClassifier .............................. 5
   3.2 AiItemsViewModel ............................. 5
   3.3 AiRecyclerViewAdapter ......................... 6
      3.3.1 Updating Active Models ................... 6
   3.4 MainActivity .................................. 7
      3.4.1 AI-Related Functionality ................... 7
      3.4.2 AR-Related Functionality ................... 8
      3.4.3 Save Functionality .......................... 9
      3.4.4 Load Functionality .......................... 9

4 Application User Preview 11
   4.1 AI Settings View .............................. 12
   4.2 AR Actions View ............................... 12

5 Experiments 14
   5.1 Testing Methodology ........................... 14
   5.2 Apparatus and Procedure ....................... 14
   5.3 Experiments: Varying AI Models ................. 15
      5.3.1 Low AI Task Intensity ....................... 15
      5.3.2 High AI Task Intensity ...................... 16
   5.4 Experiments: Varying AR Objects ............... 17
      5.4.1 Low Triangle Count ......................... 17
1 Abstract

Over the course of the research for this project, a primary task was learning and developing for a mobile, augmented reality application that utilized AI models that performed AI inferences on provided image data in order to assist in the creation, placement and rendering of virtual AR objects in the environment in real time. Specifically, the main task assigned and completed over this research time-span was to develop functionality for automated creation of predetermined configurations of AR object placements and AI task settings to provide reliable, consistent testing conditions for other experiments conducted with the application. This was successfully achieved for the most part, and the final functionality included the ability for users to save the current AI and AR configurations in the current application state, and load it at a later time with regular, timed intervals for more consistent and convenient analysis of application model and algorithmic performance. The new testing features, as a result, outmode manual testing nearly entirely. The paper goes on to exemplify the testing capabilities of the new features, examining the relationships between total AR object triangle counts, GPU utilization and AI inference throughput. There is still possibility for improved and added functionality to allow for greater varieties of experimental analyses, however the research and development conducted has left the application with a highly suitable and featured set of tools for conveniently testing the application and gathering regular resource utilization metrics.

2 Introduction

One technology that has been subject to rapidly increasing interest in the modern day is that of augmented reality (AR) — the ability to graphically project virtual objects into real-world video data in real time. From highly popular video games to educational classroom learning tools, AR has many potential applications to enhance interactivity, enjoyment and ease of use for a variety of audiences. As such, development into AR applications, particularly for mobile devices, is of relatively high priority in the worlds of computer science, computer vision and software engineering.
A specific issue of great importance is designing AR applications to work in tandem with other high intensity software that tend to rely on the same resources, such as apps using artificial intelligence (AI). Both of these technologies are typically conducted on the device GPU, and as such is important to analyze how devices handle both being run concurrently. Evaluating application response time, AI task throughput and overall GPU utilization is a meaningful objective when pursuing this analysis.

Tackling this area of research, the application in development by the team is designed to handle the creation and processing of multiple AI tasks executed by potentially different AI models on different components of the application hardware, including the GPU as well as the CPU and dedicated neural processing units (NPU). Importantly, this application is also capable of concurrently and synchronously performing both AI and AR tasks, and reporting resource utilization and performance data during runtime. The next objective, following this development, is creating additional functionality that allows users and testers to engineer their own real-time AI task and AR object configurations by saving an actual configuration of the relevant information created while using the app, and loading that same configuration later for analysis. This thesis will present this newly developed functionality and demonstrate its use in experimental analysis with a series of tests varying AI tasks and AR object configurations.
3 Application Development Overview

3.1 ImageClassifier

An abstract class used for providing classification functionality for AI tasks. Sourced from TensorFlow Lite, new TensorFlow AI models each extend this class, initializing or re-initializing fields to represent each of their different properties and behaviours. Classifiers also contain information pertaining to the AI task configuration, such as the number of threads allocated for the task, as well as the device the task runs over (CPU, GPU or NPU). While not directly modified for the purpose of this experiment, the class contained a number of vital methods for evaluating classifier performance, having methods to return response time and throughput data.

Currently, all AI models tested within the application are ImageClassifier instances. As the name describes, it is able to take in image information (in bitmap format) and inference the types of physical objects within it.

3.2 AiItemsViewModel

A class for defining a collection of fields that all correspond towards one AI task configuration. A simple class which contains an ImageClassifier object as well as fields to describe the number of threads used for the AI task and the device the task runs off of. It also contains list fields for every supported Tensorflow Lite model added to the application, as well as all possible devices usable for any particular AI task. This object is essential as a representative for managing AI tasks within the application; the automated testing scheme deals extensively with creating, updating and deleting AI tasks during a test simulation.
3.3 AiRecyclerViewAdapter

Inherits directly from the Android Platform API’s abstract `RecyclerView.Adapter` class. Android’s `RecyclerView` is a view UI element designed for users to interact with a subset of a list of some data onto the available space of the container that holds it, and the abstract `Adapter` inner class provides the functionality to bind the list of data to that particular view. Specifically, the `RecyclerView` used in the application displays and allows users to manage initialized `AiItemViewModels`, i.e. AI tasks. The `AiRecyclerViewAdapter` class implementation includes an original `updateActiveModel` method, intended to allow users to configure previously initialized AI tasks and modify the number of allotted threads, the selected image classifier and the selected device type.

Modifications to this class included the addition of a new `updateActiveMethod` overload, which isolates the core logic of the original method from the UI-related actions. The overloaded method is made public, in contrast with the original private method, so that it may be called outside the class for automated testing purposes — AI tasks can be modified without user interaction.

3.3.1 Updating Active Models

The operations conducted by the new method overload are as follows:

1. If the supplied model, device or number of threads is unchanged from the current `AiItemViewModel` fields AND the classifier is null\(^1\), return with no changes.
2. If the application’s classifier stream option is toggled on, pause the selected `AiItemViewModel`’s collector.\(^2\)
3. Update the selected `AiItemViewModel`’s fields for the model index, device index and number of threads with the supplied values.
4. Fetch the string name corresponding with the model index, and match it with the correspond-

---

\(^1\)Classifiers may be null when an AI task is first created, a non-null check allows null classifiers to become initialized even if the other settings do not change.

\(^2\)These objects also contain a `BitmapCollector` instance, which receives a feed of camera images for classification.
ing classifier instance to initialize and assign.

5. Fetch the string name corresponding with the device index, and match it with the corresponding method call to change the device.\(^3\)

6. Update the number of threads the new classifier will use with the supplied value.\(^4\)

7. Create a new collector using the new classifier.

8. If the application’s classifier stream option is toggled on, start the selected `AiItemViewModel`’s collector.

### 3.4 MainActivity

All previously mentioned components of the application culminate within the `MainActivity` class. It is responsible for setting up all of the central UI components and application logic. The relevant operations initialized and managed within `MainActivity` are outlined in the following subsections.

#### 3.4.1 AI-Related Functionality

- **AI Model List** (`mList`) — An `ArrayList` containing all created AI tasks. On application startup, a default configuration AI task with an initially null classifier is added to this list. It is also supplied to `AIRecyclerViewAdapter` as the data set.

- **AI Stream Toggle** (`switchToggleStream`) — Android `Switch` UI element that either enables or disables the input stream of bitmap data from the camera. When turned on, AI inferences can occur.

\(^3\)By default, all classifiers are set to use the CPU on initialization.

\(^4\)By default, all classifiers are set to use 1 thread on initialization.
• **Adding/Removing AI Tasks** (buttonPushAiTask/buttonPopAiTask) — Android Button

UI elements for manually controlling the number of AI tasks. Following similar behaviors, the steps for each action are as follows:

1. If the number of AI tasks is not between 1 and 20 inclusive, exit.
2. Disable the AI stream.
3. Add a new AiItemViewModel instance, or remove the last instance in the list, depending on the button.
4. Update the AiRecyclerViewAdapter with the new list as the data set, and update the recycler view with the updated adapter.

### 3.4.2 AR-Related Functionality

- **Selecting Object Names** (modelSelect) — Sets the current model that will be placed on the next placeObjectButton click event.
- **Placing Objects** (placeObjectButton) — Adds an AR object instance of the current model to the fragment window. The process for adding an object is as follows:
  - Fetch the object file corresponding with the selected current model name.
  - Create a new renderable object and add it to the list of objects.
  - Call an addObject method which calculates the exact world coordinates the new object will be placed, then places it. This is done by sending a virtual ray out from the center of the screen, and determining where the ray hits the ground.
- **Removing a Selected Object** (removeButton) — Deletes the currently selected object, i.e. the last object onscreen the user tapped on, from the list of rendered objects.
- **Clearing All Objects** (clearButton) — Clears the rendered object list entirely.
3.4.3 Save Functionality

One of the main contributions towards testing for the purposes of this thesis are the ability to save the current application AI task and AR scenario state to a file. The functionality is straightforward; the function reads the current AI model list and renderable object list fields, and saves the following information from each element of those lists:

- **Per AI Task** (*AiItemsViewModel*) — number of threads, AI model (image classifier), device
- **Per AR Object** (*baseRenderable*) — Object model name corresponding with object (.sfb) file name, x-offset of object center from screen center in pixels, y-offset of object center from screen center in pixels

A new save generates a new folder in the `saved_scenarios_configs` subdirectory within the application data folder. All folders are numbered as `saveN`, describing the number of saved scenarios within the directory. Within the folder are two CSV files for task configuration and scenario settings respectively, holding records of information for each AI task and AR object model saved. Finally, saves are automatically reloadable after being created using the `scenario` and `taskConfig` dropdown menus as described in the following section.

3.4.4 Load Functionality

The other main contribution to application functionality, and the one most relevant for testing the application, is the ability to load previously saved AI task configurations and object scenarios, either in the current runtime or any previous runtime. This is achieved by accessing the `saved_scenarios_configs` subdirectory and loading the corresponding filenames into the `scenario` and `taskConfig` dropdown menus present in the AR Actions application view.
The process of loading a saved task configuration and scenario is as follows:

1. Clear all current objects from the screen, and empty the AI model list so no AI tasks are active.
2. Read the selected filenames from the `scenario` and `taskConfig` dropdown menus, and load the respective CSV files for reading.
3. For every record in the task config CSV, create a new `AiItemsViewModel` object with the corresponding field values and add it to the AI model list on a fixed-rate timer.⁵
4. Once all AI tasks have been created, turn on the AI collector stream.
5. For every record in the scenario CSV, create a new renderable object from the given model name, place at the recorded position on the screen, and add it to the renderable object list on a fixed-rate timer.⁶
6. Once all objects have been placed on the screen, begin removing each object in reverse creation order on a fixed-rate timer.⁷

⁵Timer tick length is defined by the `taskConfigTickLength` field in `MainActivity`.
⁶Timer tick length is defined by the `scenarioTickLength` field in `MainActivity`.
⁷The timer tick length is shared between the scenario creation and removal timers.
4 Application User Preview

(a) AI settings
(b) AR actions

Figure 1: The main modes of the application
4.1 AI Settings View

1. **Fragment View** — The image of the real world captured by the camera. It sits behind the rest of the UI.

2. **AR Object** — A placed AR object. By default, objects are placed at the center of the screen (seen as a green dot).

3. **AI Visibility Toggle** — Toggles the AI settings recycler view.

4. **Push/Pop AI Tasks** — Adds or removes an AI task to or from the end of the AI model list.

5. **Toggle AI Stream** — Enables or disabled the data stream for AI inferences.

6. **AI Settings Recycler View** — Contains all AI control functionality. Scroll left or right to view settings for each active AI task.

7. **AI Model Properties** — A quick textual description of an AI model in use.

8. **Thread Count Selection** — A scrollable list view of thread count options. The AI model is updated automatically once the user stops scrolling.

9. **Classifier Selection** — A scrollable list view of classifier options. The AI model is updated when the user taps on a list item.

10. **Device Selection** — A list view of device options. The used device is updated when the user taps on a list item.

4.2 AR Actions View

11. **Model Selection** — A dropdown menu of available model options, with the current model visible onscreen.

12. **Place Object Button** — Places an AR object of the current model seen in 11.

13. **Remove Object Button** — Removes the currently selected AR object. The selected object has a white circle underneath it.

14. **Clear Button** — Removes all objects from the screen.

15. **Save Configuration Button** — Saves all current AI task configurations and AR scenarios
to a new subfolder in local storage.

16. **Load Configuration Button** — Automatically loads an AI task configuration and AR scenario from local storage. The configs selected are shown in 17 and 18.

17. **AR Object Config Selection** — A dropdown menu of available AR scenarios, with the current scenario visible onscreen.

18. **AI Task Config Selection** — A dropdown menu of available AI task configs, with the current config visible onscreen.

19. **Automatic Decimation Button** — Decimates onscreen objects on a timed interval. *Not used in experiments for this thesis.*

20. **Automatic Decimation Button** — A slider showing the current decimation percentage. Sliding right means more decimation. *Not used in experiments for this thesis.*
5 Experiments

5.1 Testing Methodology

The new testing functionality as described in the Application Development Overview was utilized to conduct a series of simple experiments to analyze the performance of the application. Specifically, combinations of different AI task configurations and AR scenario were loaded into the application during runtime, and metrics such as GPU utilization and AI throughput were collected on timed intervals. With this, it is possible to view and record the performances of these different models relative to one another and confirm that they align with testing and real-world expectations.

Four total experiments were conducted as demonstrations of the automated testing functionality. Half were focused on analyzing the behavioral differences between a light and heavy AI task configuration with respect to GPU usage, AI throughput and total triangle count. The other half were designed to help analyze how different scenarios are handled by the same AI task configuration, analyzing the trends of the same resources’ utilization metrics as the previous test type with denser or sparser triangle counts. Different image classifiers were selected for the low- and high-intensity based on metrics obtained from TensorFlow Lite [1].

5.2 Apparatus and Procedure

All testing was conducted using a OnePlus 5 suspended with a phone stand, camera raised to a height of 1 metre from the ground and inclined at 40 degrees to the horizontal. This results in roughly 1.3 metres of distance between the camera and the point ground it focuses on.

To perform a test, an AI task configuration and scenario were selected using the AR actions view as seen in the Application User Preview, then the Load button was pressed. When a test is completed, the application is paused and the collected data is copied and saved.
5.3 Experiments: Varying AI Models

5.3.1 Low AI Task Intensity

Observations

The low intensity AI task experiment presented with a baseline GPU utilization of around 55-60%. When the AI collector stream was enabled, the GPU baseline rose to 70-75%, and the throughput baseline was 35-40 completed inferences per second. As each plant was added, the GPU utilization steadily rose and peaked shortly before the last few objects were placed. Throughput decreased as the objects were added with a low of 17-20. In the removal phase both GPU utilization and throughput regained baseline performances after the AI stream was enabled.

Notably in Trial 2, the total triangles wavers slightly and does not make a perfect peak in the plot. This is due to object decimation, where the application automatically merges triangles in the object mesh together to reduce computational strain.
5.3.2 High AI Task Intensity

Observations

This experiment presented with a baseline GPU utilization of around 55-60%, but immediately shot up to over 90% utilization when the stream was enabled. Throughput was incredibly low at base as well, slowly dropping from 4 to 2 over the course of the experiment.

No object decimation occurred in this experiment interestingly. This may be in part due to the model simply not supporting the functionality, or perhaps the utilization was so high from the start of the stream that decimation would be ineffectual.
5.4 Experiments: Varying AR Objects

5.4.1 Low Triangle Count

Observations

This experiment was designed to examine the behavior of the application and its resource utilization in situations where max GPU utilization is not reached. With a moderately capable image classifier and extremely simple models relative to the control scenario, resource utilization was steady and low. With a typical GPU baseline of 55-60% before the AI stream and 70-75% after, GPU utilization peaked in the low 80s. Throughput utilization started at 40-45 and barely dropped to about 35. Interestingly, throughput in Trial 1 continued to trend downwards even after objects had been removed. No object decimation occurred.
5.4.2 High Triangle Count

Figure 5: Trials for Low Triangle Count (MobileNet V2 Float, 1 thread, GPU | 9 plant, 9 Cocaca-laFinal, 9 spline)

Observations

Compared to the low triangle count experiment, this experiment produced a variety of interesting results. GPU utilization had a standard baseline before and after the AI stream, and was steadily rising to 100% utilization. Throughput had a recognizable baseline of around 40, and slowly dropped over time as objects were placed. Object decimation occurred relatively early, however, and had a significant, highly visible impact in the utilization metrics. GPU utilization went from over 90% to suddenly hovering anywhere from the 60s to the 80s. Throughput bowed upwards from its steady descent, but still met a low of around 15. Object decimation was prevalent for the rest of the trials after it had begun.
6 Results and Discussion

The results obtained from the conducted experiments largely correlated with the expectations derived from their TensorFlow Lite descriptions. The low-intensity MobileNet V1 Float image classifier initialized with the AI collector stream had relatively low initial resource consumption, sitting around 65-75% GPU utilization, and generally took longer to reach maximum GPU utilization in the tests. It was also observed that the lower-end MobileNet models tended to defer to object decimation under pressure of extreme GPU load (heavy scenarios), resulting in sudden and massive drops in triangle counts that eventually relaxed once the load was reduced. Throughput metrics also started quite strongly, with values in the 40s, but dropping as low as the single digits at max load. For lighter scenarios, no object decimation was observed, but throughput data had higher variance.

Conversely, the heavier Inception V4 classifier had extremely high GPU utilization the moment the stream was started, beginning at around 90% GPU utilization. Notably, no object decimation was noticed when the Inception V4 classifier was used. It could be theorized that the model had initially high GPU consumption and low throughput because of some sort of optimized batch processing, utilizing more memory at one time to more efficiently process image data.

Generally, it can be seen that the automated testing tools developed within the research project adequately presented expected behaviour from the selected image classifiers for a variety of different combinations with most phenomena being predictable and adequately presented in the collected data.
7 Limitations

While the experiments were largely successful in demonstrating the efficacy and validity of the automated testing tools, it is certain that there were some number of flaws and drawbacks to the demonstrated testing procedure and methodology. The primary weak point of the experiments was the utilized hardware. The tests, as stated in the Experiments section, were conducted on a OnePlus 5, which at the time of this thesis is about 5 years old. Numbers of advancements in the tested image classifiers have been made in the years since, and the relatively minimal AI task configurations used in the experiment were in part necessitated by the lower power of the device. Thermal issues were fairly common, and had to be controlled with a cooling fan. This did leave an impact on the data, as resource consumption took noticeably longer to recuperate than to rise in some tests.

8 Next Steps

An obvious and important fact is that there are an incredible number of variables and possible contexts under which a mobile AR application would be expected to run and an AI image classifier application would be expected to perform inferences on. The tests conducted in this thesis, while demonstrative, are not comprehensive, and a number of additional experiments can be designed to test different lighting conditions, different real-life object placements in view of the camera and other attributes of a testing environment. Data collection and analysis on response time, number of executed tasks and possibly relationships between time, AI/AR configurations, resource utilization metrics and temperature can be examined.
9 Conclusion

In summary, the combination of nuanced augmented reality and artificial intelligence functionality within the team’s application necessitated the creation of a robust and simplistic testing feature that allows users to save, load and analyze all kinds of AI task configurations and AR object placements to determine the efficacy of the utilized image classifiers as well as the application as a whole. The testing functionality needed to be able to show the projected behaviour of the selected AI models alongside certain Ar object scenarios and accurately demonstrate the roughly direct correlation between total triangle count and GPU utilization, and the inverse relationship between total triangle count and AI throughput. The initial application was able to demonstrate this through manual human testing, which was prone to a variety of potential human errors. By the end of development for this research, the new functionality effectively improves upon all possible testing in terms of consistency and easy of use. With the endless number of experiments on this application and the technology utilized, the developed testing tools will serve as an effective and flexible method for any context conceivable.

Acknowledgements

The author would like to deeply thank Dr. Marco Brocanelli and Niloofar Didar, who lead current research and development on the application, and Joseph Bommarito, who was incredibly helpful and supportive throughout the research and development for this thesis.
References


Appendix

Creating AI Task Configurations and AR Object Scenarios

While it is possible to save and reload configurations and scenarios while using the app, it is also possible and recommended for dedicated experiments to generate CSVs for them by hand.

Example format for `taskconfigN.csv`:

```
threads,aimodel,device
1,Mobilenet v2 Float,GPU
```

Example format for `scenarioN.csv`:

```
model,xOffset,yOffset
hammer,-300,-300
hammer,-300,0
hammer,-300,300
plant,0,-300
plant,0,0
plant,0,300
apricot,300,-300
apricot,300,0
apricot,300,300
hammer,-300,-300
hammer,-300,0
hammer,-300,300
plant,0,-300
plant,0,0
plant,0,300
apricot,300,-300
apricot,300,0
apricot,300,300
```
Testing Screenshots

Low Intensity AI Task
High Intensity AI Task
Low Triangle Count
High Triangle Count

![Image of a device screen showing a high triangle count with a count of 27.]