Lab 3: Mobile Deep Inference

Total possible points: 890 Points
Bonus points: 200 points
Minimum points required: 200 points

This project is designed to help you understand the overall process of using deep learning models in a mobile application. This project has three parts demonstrating the logical process of extending an open-source project. You could attempt Part 1 and Part 2 in any order, but the completion of Part 3 will depend on the previous two parts. Upon successfully completing this project, the mobile app will have a new feature that allows the use of cloud-based inference.

For all parts, it is your responsibilities to demonstrate to the teaching staff the successful completion of the requirements, e.g., by including the requested deliverables, clearly demonstrating and explaining in the screen recording, voice over, and README the completion, to earn points for each rubric item.

Part 1: On-Device Inference (200 Points)

Many real-world mobile applications now leverage deep learning models to provide novel features. A canonical example is leveraging image classification models, such as EfficientNet or ResNet model families, to recognize objects in each image automatically. Photo gallery apps can then allow users to query those images using predicted labels.

In this part, we will look at a simplified Android project [1]that shows how to run image classification inference tasks using Google's on-device inference framework called LiteRT. At a high level, this Android app allows the user to run an image classification model on each camera frame with the desired setting, e.g., which model to use and how many results to display.

Your main task for this part is understanding how the on-device inference is implemented. The starter code for this part can be found inside the Part 3 directory. Later in Part 3, you will leverage this understanding to figure out how to implement the cloud-based inference feature. The ability to work with open-source projects is a critical skill set because it opens the door for further development.

[1] The starter code is based on: https://github.com/google-ai-edge/litert-samples To make this lab more manageable, I modified the original image classification app to have fewer features and less codebase so you can focus on the key concepts of on-device and cloud inference. Interested students can check out the original code as linked above.

To demonstrate your understanding, you need to answer the following questions.

- How was the camera preview implemented?
- On the bottom sheet, the user can change the settings, e.g., choose a different EfficientNet model while using the app, and the app will start using the new setting. How was this feature implemented?
- How did this app specify which deep learning models to be included in the apk, and where are these two deep learning models stored?
- What is the data and control flow to run inference on each camera frame and display the result on the bottom sheet?

Getting started can be hard if you haven't had much experience working on complex real-world projects. However, the end will be rewarding. Below, we provide some suggestions and tips.

- Use the Android official document to understand the key programming concepts. In this project, a basic and conceptual understanding of how Compose, ViewModel, and StateFlow work will be useful.
- Spend some time to interact with the app. Understanding the app's behaviors from the end-user's perspective will help you establish the correct mental model of what to expect when inspecting the code.
- Use Logcat to add debug information to help orient yourself regarding the key workflows.
- Leverage the IDE's debugger as much as possible to understand specific code block behaviors.

Deliverables

For this part, you should submit the following files and place them in the Part I folder.

- A file named `part1` consisting of your responses to the above questions. Please write your responses in prose and keep them concise. You can use diagrams or code snippets where appropriate to help you better answer the questions. Though refrain from copying and pasting large chunks of codes directly from the project, as they do not demonstrate understanding.

Gradings

Requirement	Brief Description	Points total
On-device inference Q1	Clearly explained and correctly answered?	50
On-device inference Q2	Clearly explained and correctly answered?	50
On-device inference Q3	Clearly explained and correctly answered?	50
On-device inference Q4	Clearly explained and correctly answered?	50

Helpful Hints: if you have trouble directly starting with the starter code, perhaps because you need to spend time understanding Android programming concepts, you could try the alternative rubric listed below. Although you can't get more points for completing this part, I hope this alternative path can help get you started and reward you points for your time and effort.

The Jetpack Compose basics: https://developer.android.com/codelabs/jetpack-compose-basics#0
The LitRT documentation: https://ai.google.dev/edge/litert;
https://ai.google.dev/edge/litert/android/index

Requirement	Brief Description	Points total
The Jetpack Compose basics Codelab	Clearly demonstrating the completion of the Codelab	30
The LitRT documentation	Clearly demonstrating reading the documentation	10
On-device inference Q1	Clearly explained and correctly answered?	40
On-device inference Q2	Clearly explained and correctly answered?	40
On-device inference Q3	Clearly explained and correctly answered?	40
On-device inference Q4	Clearly explained and correctly answered?	40

Part 2: Cloud-Based Inference Server (110 Points)

In this part, you will implement a simple inference server that can host one image classification model and return the top five prediction results. Each prediction result is defined as the predicted class label and the classification probability.

We provide a simple flask-based inference server that can return the top prediction result. You are welcome to extend it or write your own from scratch. To quickly test this server, you can use `curl` as shown below. This command will send the test image to the inference server that listens on port 5050 on localhost and return the prediction result.

```
tian@Tians-Mac-Studio inference_server_project-reference % curl -X POST -F "file=@test-images/dog.jpg"
 http://127.0.0.1:5050/predict

{"predicted_label":"Doberman","score":0.8287531733512878}
```

After completing this part, your inference server should return the top-5 results. An example output is shown below.

```
tian@Tians-Mac-Studio inference_server_project-reference % curl -X POST -F "file=@test-images/dog.jpg"
 http://127.0.0.1:5050/predict

{"predictions":[{"predicted_label":"Doberman","score":0.9957420229911804},{"predicted_label":"miniatur
e pinscher","score":0.0017673427937552333},{"predicted_label":"toy terrier","score":0.0003933749976567
924},{"predicted_label":"Italian greyhound","score":0.0003056807618122548},{"predicted_label":"Weimara
ner","score":0.0002079797995975241}]}
```

We recommend you also look at the client.py in the inference_server_project-starter directory. Currently, this simple client can parse the top 1 prediction result. This Python client provides you with the basic workflow for constructing an HTTP request and handling the JSON response. Try to modify the client so it can correctly parse the top-5 prediction results. Afterward, you will be fully prepared to move on to Part 3, where you will implement a similar client-side logic in Android.

Deliverables

For this part, you should keep the starter code directory structure in Part 2 and include the following files:

1. The `image_classification_server.py` file. Regardless of whether you directly modify the starter code or write your own server, you should place your server code inside this file.
2. If your server requires additional packages, you should update the README to include that information.
3. Additionally, include two screenshots, one for starting the server and one for displaying the curl response, that demonstrate the completion of the requirements.

Gradings

Requirement	**Brief Description**	**Points total**
Inference server code	Server starts successful?	40
Top-5 classification results code	Curl returns the prediction results?	50
Screenshot 1	Server starts successful?	5
Screenshot 2	Curl returns the prediction results?	5
README	Clear?	10

Part 3: Supporting Cloud-Based Inference (380 Points)

In this part, you will extend the image classification Android app you studied in Part 1 to allow users to offload image classification tasks to a remote server. In practice, this remote server is hosted at the edge or in the cloud. For simplicity, in this project, you can use the simple inference server you set up in Part 2.

There are different ways you can successfully complete this part. To help you get started, we provide the logical steps I took to implement the cloud-based inference feature. You will have to fill in some of the details yourself by looking at different libraries' documentations.

1. Start by implementing the function that mirrors what the Python client does. That is, the function will create an HTTP request, send it to your inference server, and parse the response. You can use any HTTP client libraries for Android. Both okhttp and Retrofit are popular and easy to use. To handle JSON, one option is to use Android's official package org.json.
2. You will need to figure out where to place this function and how to trigger this function from the UI. But before that, we recommend you test this function's implementation first. One simple way to test is to slightly modify the control flow to always call this cloud-inference function. If the function is implemented correctly, you should see the server is getting the POST request at /predict and the response is successful with status code 200; the Android app should display slightly different predicted labels and inference time than when running on-device inference with the EfficientNet models.
3. Lastly, you will modify the UI with some new composable function(s) and update the UI state properly so that different inference modes trigger different execution paths. Your UI does not have to match the reference implementation exactly. But your UI should allow the user to choose to run inference requests either on the device or in the cloud; and when the cloud option is selected, on-device settings should either be disabled or hidden.

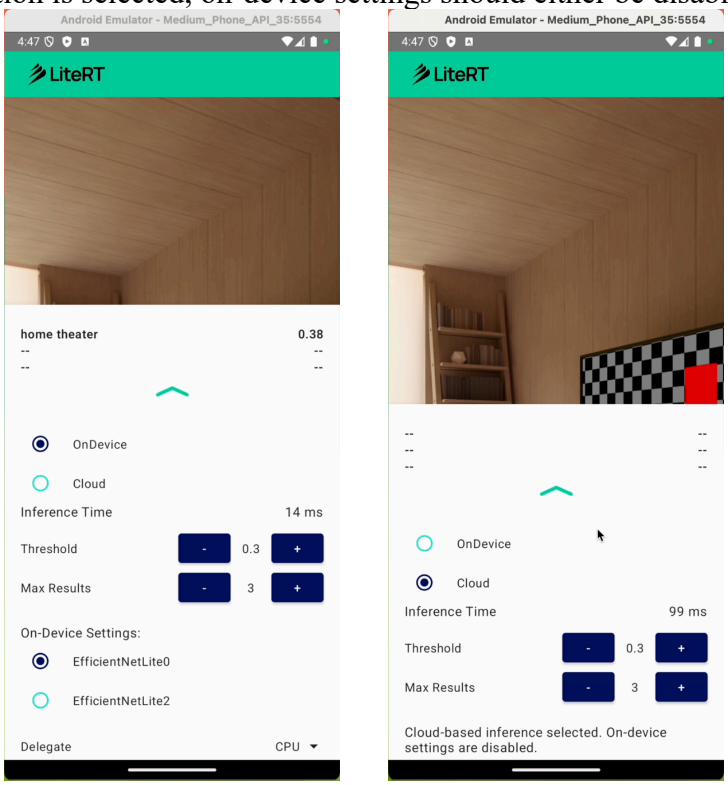

Figure 1: Screenshots of reference implementation that allows the user to send the image classification requests to a separate server.

You will likely encounter some Android-specific peculiarities, e.g., regarding permissions or plaintext connection. You should be able to figure them out quickly by looking up how to modify `AndroidManifest.xml`.

Unfortunately, it is likely some things will break between when I wrote this lab and when you will be doing the project, as Android constantly changes things. So, if you get stuck for an unreasonably long time, please feel free to post on Canvas. This is part of the fun of working on Android, so try to enjoy or commiserate with me.

Deliverables

For this part, you should submit the following deliverables in the Part 3 folder:

- **The debug apk with the name cloud.apk**. You can do so in Android Studio by following the screenshot below. Afterwards, the debug apk can be located in the app/build/outputs/apk/debug directory.
- **The source code of your app**. Please do not include build-related files in the submission, which can significantly increase the submission size. Similarly, you can do so in Android Studio by following the screenshot below with the *Clean Project* option.
- **A screen recording of the implemented features.** This recording should include your voice-over briefly explaining to the teaching staff how the implementation satisfies the feature requirements. Physical Android phones have a built-in screen recorder that can record both the screen and microphone. If you are using Android virtual device (AVD)[2], you could use and screen recording apps from your laptop. Please keep the recording as concise as possible.
- **A README file.** You need to include your team information (name and email address) and brief explanations of how you implement the required app features. Only plaintext writeups are accepted, and markdown files are also okay.

Gradings

The prerequisite for earning the points for apk, source code, screen recording, and voice over is to have a working prototype for cloud-based inference.

Requirement	Brief Description	Points total
apk	Included?	10
source code	Reasonable?	10
Screen recording	Clear?	20
Voice over	Clear?	10
Cloud-based inference UI	Reasonable?	80
Cloud-based inference function	Complete?	200
README	Clear?	30

[2] As of 2025D, Android Studio seems to remove the support of enabling audio-recording when using AVD.

Bonus Part (200 Points)

Deep learning (DL) has seen great adoption in mobile and ubiquitous computing. This lab so far introduces you to one of the earliest DL tasks (image classification) that see commercial adoption in mobile devices.

There are so many cool advancements, e.g., on-device LLM, that you can explore! So, in this part, I will give you the opportunity to define what you think are cool. To earn the points for this part, you just need to spend time to understand and gain hands-on experience on a specific topic related to **mobile deep learning**. Even better, you also get to tell us what the deliverables are and how points should be assigned.

Here is how it works:

1. Identify a specific topic you would like to spend time working on. This topic should have sufficient technical depth and require non-trivial amount of time to understand. If you need some pointers, see below a suggested list.
2. In a document called `Bonus`, write down the following three information:
    a. A brief description about the problem and the learning objectives.
    b. What type of deliverables should be expected upon successful completion of the problem.
    c. A grading table that specifies how points should be distributed to different deliverables.
3. From this step on, proceed normal as how you would for any other parts.

Because you get to decide what to do, and what counts as success, please clearly communicate this information in the `Bonus` document.

Here is a list of suggested topics.

1. Interested in mobile programming for iOS? Learn how to implement image classification in iOS: https://github.com/google-ai-edge/litert-samples/tree/main/examples/image_classification/ios
2. Interested in other DL tasks or modalities than the image? Learn about other tasks in the LiteRT samples: https://github.com/google-ai-edge/litert-samples/tree/main/examples
3. Curious about LLM? Learn how to implement an on-device LLMs: https://github.com/google-ai-edge/mediapipe-samples/tree/main/examples/llm_inference/android
4. Heard of MediaPipe and always want to try it out? Start here: https://ai.google.dev/edge/mediapipe/solutions/guide

Finally, you are not limited to iOS or Android, you can also choose to work with cross-platform development option for this part. You are not limited to Google's on-device AI solutions. The key requirement for this part is to pick a topic related to mobile deep learning, and have fun!

Deliverables

For this part, you should submit the following deliverables in the Part Bonus folder:

- **The `Bonus` documentation.** Follow similar formats as previous project description to describe what needs to be done, the deliverables, and a grading rubric to assign points.
- **Deliverables as requested in the `Bonus` document.**
- **A README file.** You need to include your team information (name and email address) and brief explanations of how your project design has sufficient technical depth and whether you have successfully completed your own project. Only plaintext writeups are accepted, and markdown files are also okay.

Gradings

We will follow the grading rubric you created to assign points to each part, if they are reasonable.

Checkpoint Contributions

Students must submit work that demonstrates substantial progress towards completing the project on the checkpoint date. Substantial progress is judged at the discretion of the grader to allow students flexibility in prioritizing their efforts. For this assignment, completion of either Part 1 or Part 2 will be considered as making substantial progress. Projects that successfully submit a checkpoint demonstrating significant progress will receive 10 points.

Errata

Report errors by emailing tian@wpi.edu with the subject line *[CS4518] Lab [Number]*. Thank you for taking the time to help us improve our courses.