EMBEDDED SYSTEMS WORKSHOP

GESTURE BASED TEXT EDITOR ON EDGE

Team: Alt + F4

Ayush Daga, Siddarth Meda, Karthik Sundram, Abdul Ahad Shoeb

Problem Statement

To develop a gesture-recognition smart interface on the Qualcomm Innovators Development Kit (QIDK) leveraging its AI-on-edge features (Qualcomm AI stack) to optimise the performance.

Motivation

Among the primary reasons for us selecting this topic was that gesture based text editors can help in mixed-environments where there are hard of hearing people among others as it is hard for everyone to learn sign language due to its complexity, however learning a small number of signs is very easy to become fluent in.

Aim

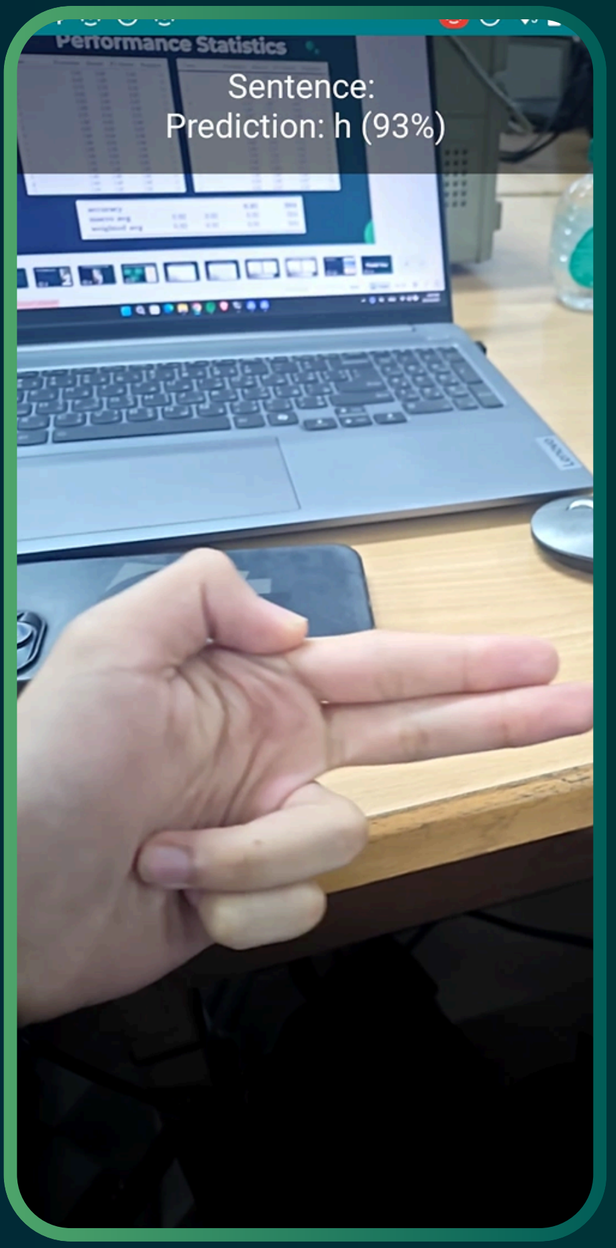
1. To create a low-latency app that detects the signed letters and symbols and accurately displays the text signed

2. Try and optimise the performance of the trained AI model for deployment on the QIDK

3. Compare the performance of the model on a variety of processors

Our Progress so Far

- We have a working Android application that runs inferences on live captured frames.
- The application utilizes CameraX for camera access and a MediaPipe-based model for real-time gesture recognition.
- The app flashes a light to signal the signer to move on to the next sign while it displays the recognised sign in the live text display.
- We have successfully got the app to run on the QIDK’s CPU and GPU.
- We also have implemented an autocorrecting algorithm using edit distance and dictionary lookup in case of occasional signing and/or inference mistakes.



App Functionalities

- On app start-up, the app initializes and first checks for camera permissions.
- Starts using the rear camera and sends each frame detected through the inference pipeline, which predicts a gesture with a certain confidence level.
- The app checks if the same gesture with high confidence is detected for 15 frames continuously, and then appends the character to a buffer.
- Once the character is confirmed, it flashes the rear camera’s light to tell the signer that the gesture has been captured successfully.
- This is followed by a short cooldown of 0.5 seconds before getting ready to scan the next gesture.

App Functionalities

- We tried other methods to maximise accuracy of the text displayed such as finding the sign with maximum confidence over a window and the most frequently inferred sign over a window but they gave lower accuracy than the method we currently adopted.

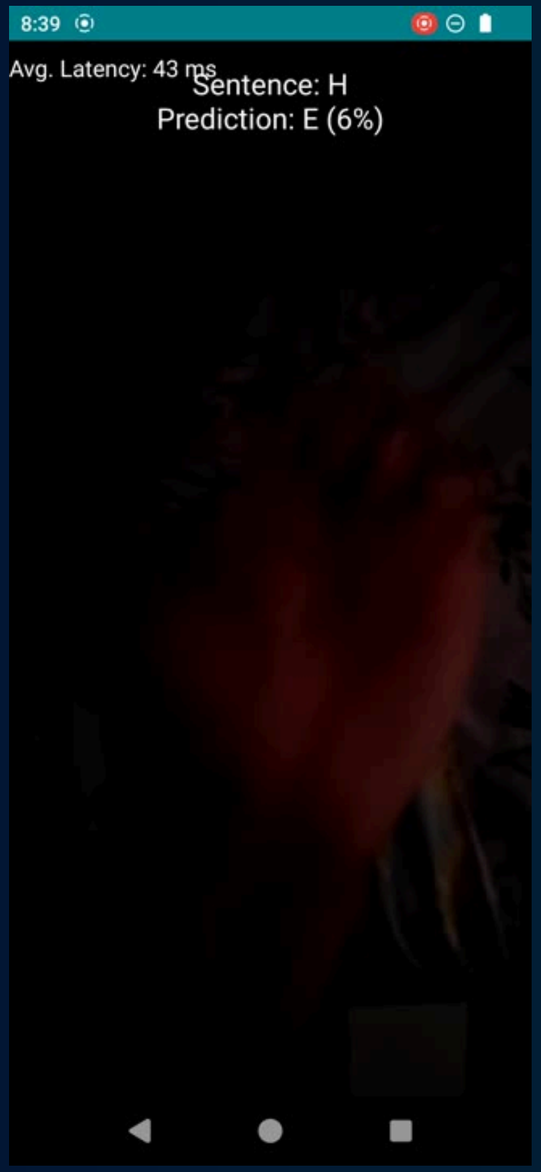
- Once a signer signs a space, we run an autocorrect algorithm to correct possible inference or signing errors so the text displayed is coherent.

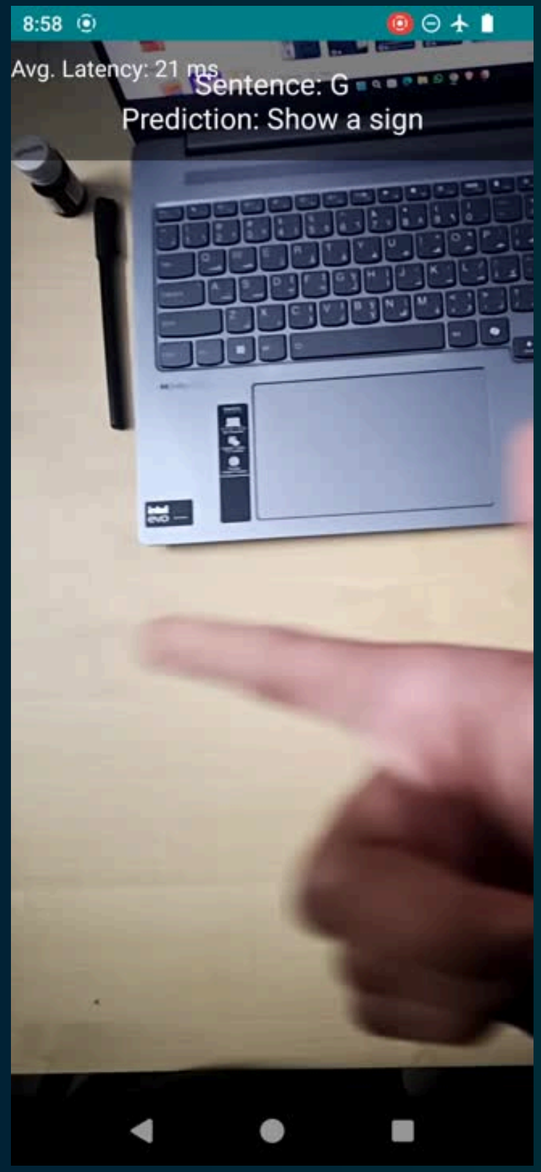
- The autocorrect algorithm is based on Levenshtein distance which finds the most probable word with least “difference” from a dictionary of 300,000 word-occurrence pairs.

- Two spaces signed in a row results in a full stop and three spaces in a row clears the text displayed on the app screen.

App Demonstration

- Model works quite well in dim lighting as well.





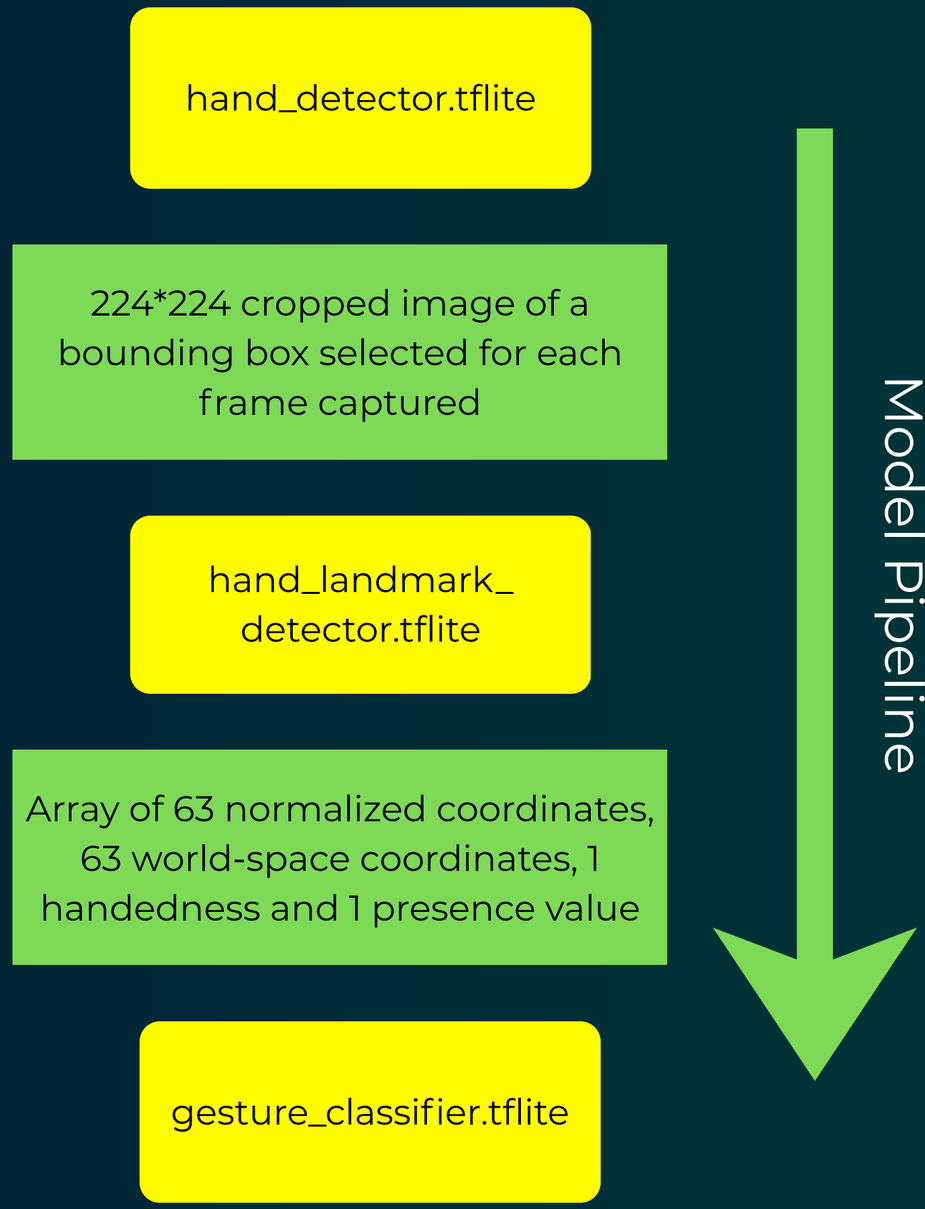
- Model is trained on both hands and thus is unbiased to orientation.

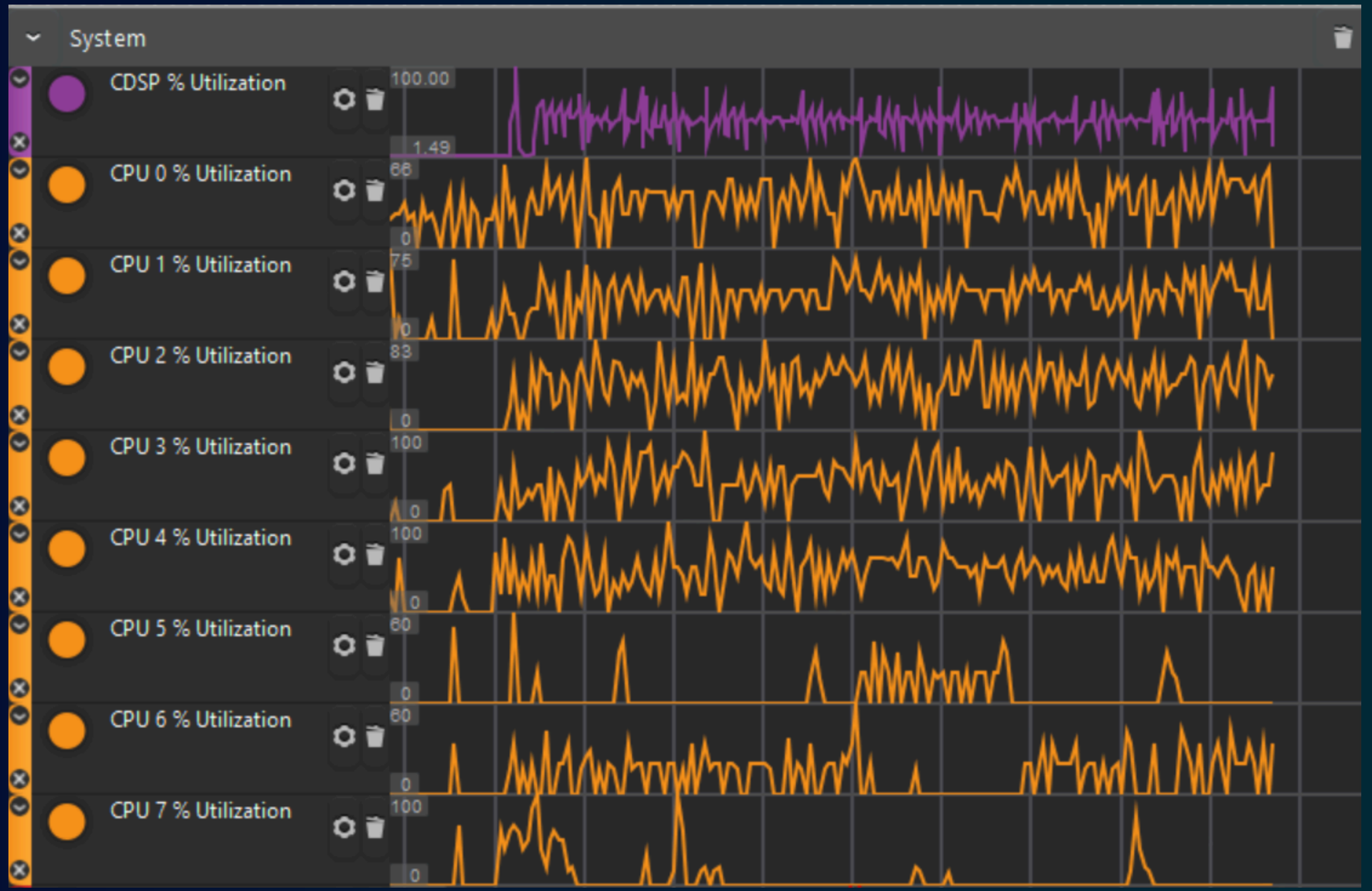
Attempts at NPU Execution

- We were initially using MediaPipe which conveniently bundled multiple .tflite files into one .task file.

- MediaPipe cannot delegate to the NPU. Hence we had to write our own pipeline code to replace MediaPipe.

- Successfully runs on NPU but output is wrong due to remaining issues with the pipeline.

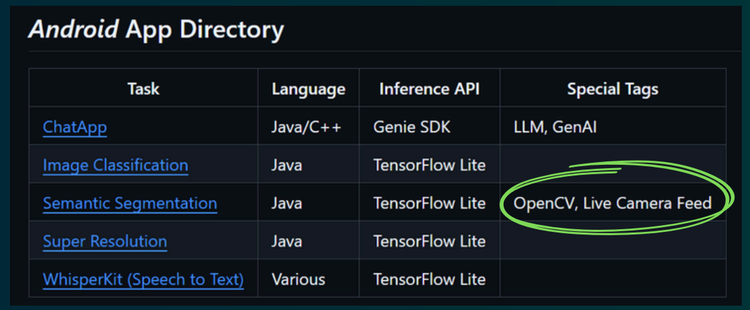




CDSP usage is there but model gives very wrong outputs.

Another Attempt at NPU

- We found a Qualcomm GitHub repository “Qualcomm AI Hub Apps” with many example apps with models running on the NPU.
- One of the examples matched our case: OpenCV, Live Camera Feed so we tried to use this app as a basis for our app.
- However, the problem is that this app uses only one model and hence effectively uses the NPU. While, we have multiple models bunched up together in a .tflite.
- So we tried to use a very large dataset (3,000 images per class) to train a singular CV model instead of using MediaPipe model maker using code we got from a research paper.
- But even with such a large dataset, the accuracy of the model generated was very poor (34%).



Android **App Directory**

Task	Language	Inference API	Special Tags
ChatApp	Java/C++	Genie SDK	LLM, GenAI
Image Classification	Java	TensorFlow Lite	
Semantic Segmentation	Java	TensorFlow Lite	OpenCV, Live Camera Feed
Super Resolution	Java	TensorFlow Lite	
WhisperKit (Speech to Text)	Various	TensorFlow Lite	

AI Model Requirements

Model Runtime Formats

- TensorFlow Lite (.tflite)

I/O Specification

INPUT	Description	Shape	Data Type
Image	An RGB image	[1, Height, Width, 3]	float32 input expecting inputs normalized to [0, 1]

OUTPUT	Description	Shape	Data Type
Classes	**CityScapes** Classes	[1, Height', Width', 19]	float32 lower resolution class logit predictions

Refer to the CityScapes segmentation model.py for class label information.

The app is developed to work best with a Width/Height ratio of 2.

Final Attempt at NPU

- First, we exported our trained TfLite model to ONNX and validated it with ONNX Runtime to ensure functional correctness and operator compatibility.

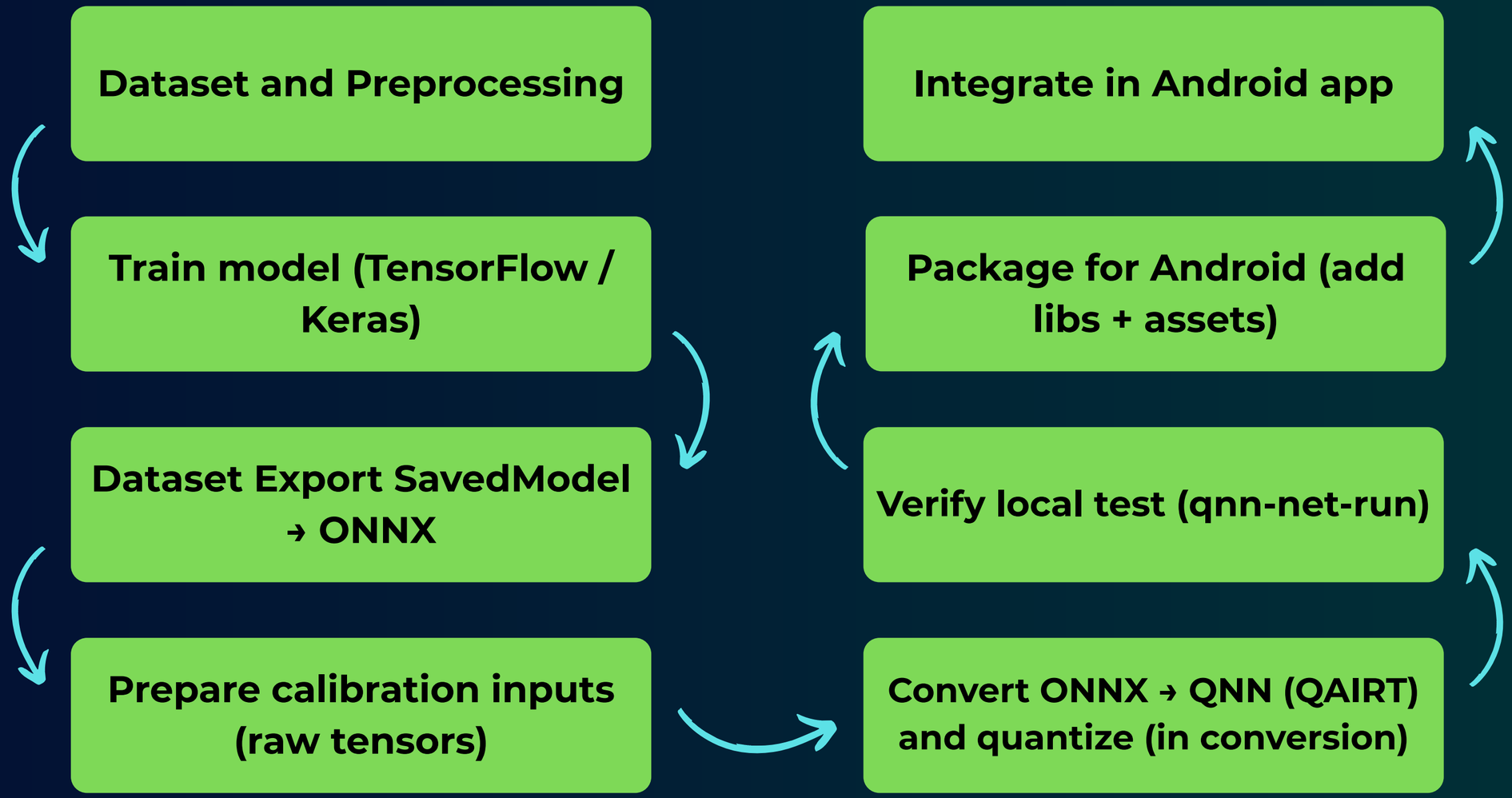
- Then we converted the ONNX model using QNN’s ONNX converter to generate QNN-compatible model files and identify any unsupported layers.

- Next, we performed INT8 quantization with calibration data to target NPU acceleration and reduce model compute + memory cost.

- Finally we built the NPU execution binaries using the QNN Model Builder, producing deployable backend-specific artifacts.

- But at the end, when we tried to integrate it with our Android app, we faced a lot of errors such as the absence of QNN Backend libraries on the device. We tried to add them manually, however we couldn’t push QNN libraries directly to the app due to persisting permission issues. Also, QNNSetupBackend was failing.

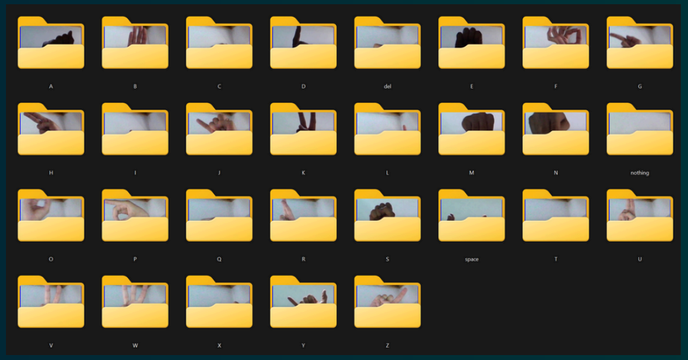
Training and Quantisation Pipeline

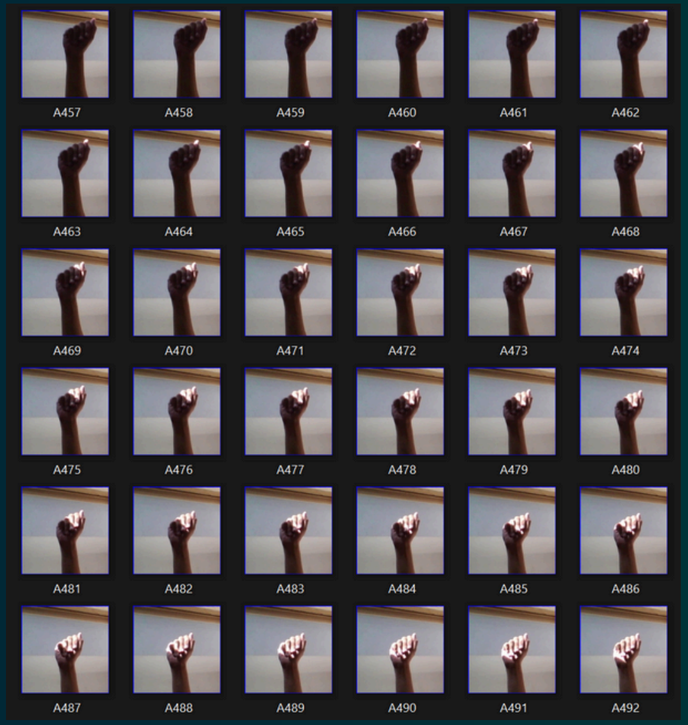


Methodology

Dataset

- The dataset we used contains 29 classes: 26 alphabets, nothing, space and delete.
- It contains 3,000 images per class with variations in lighting, hand distance and signers.
- The dataset was divided into test, train and val as required to calculate performance statistics.
- We repurposed some sequences to map to some special actions such as full stop and clear all.





Model Training and Performance

F1 Score

0.99

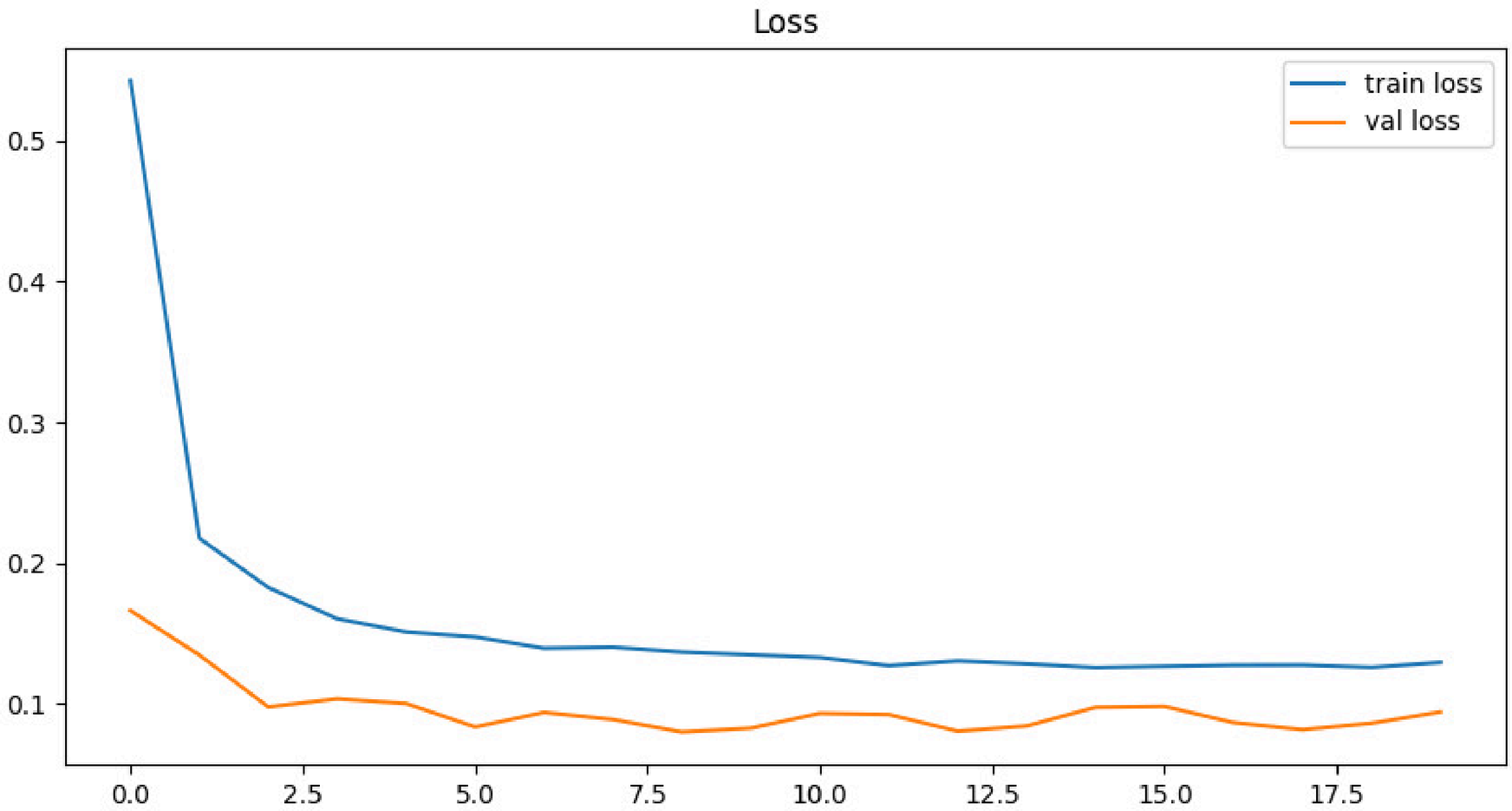
Accuracy

0.9952

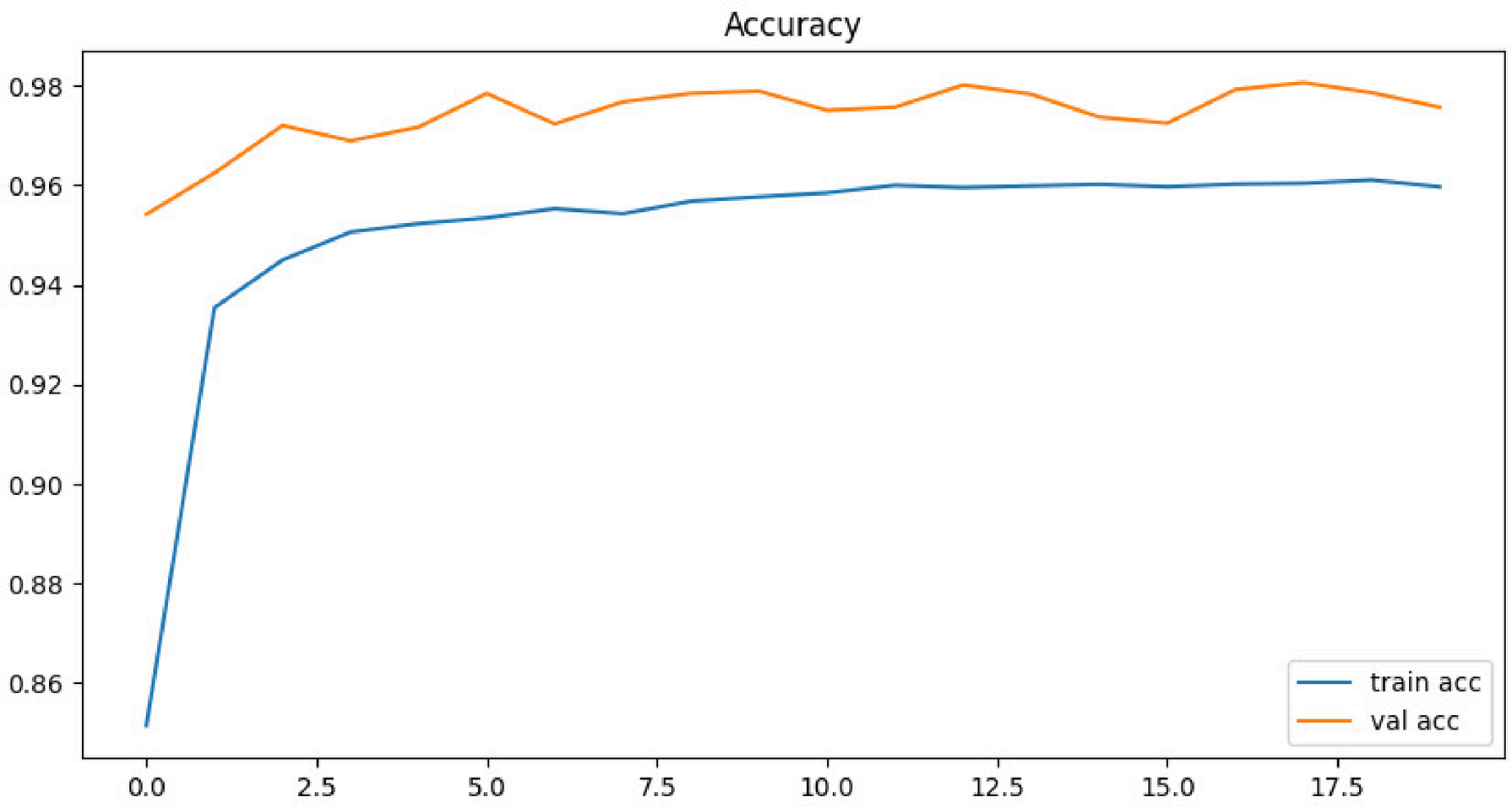
Loss Rate

0.18

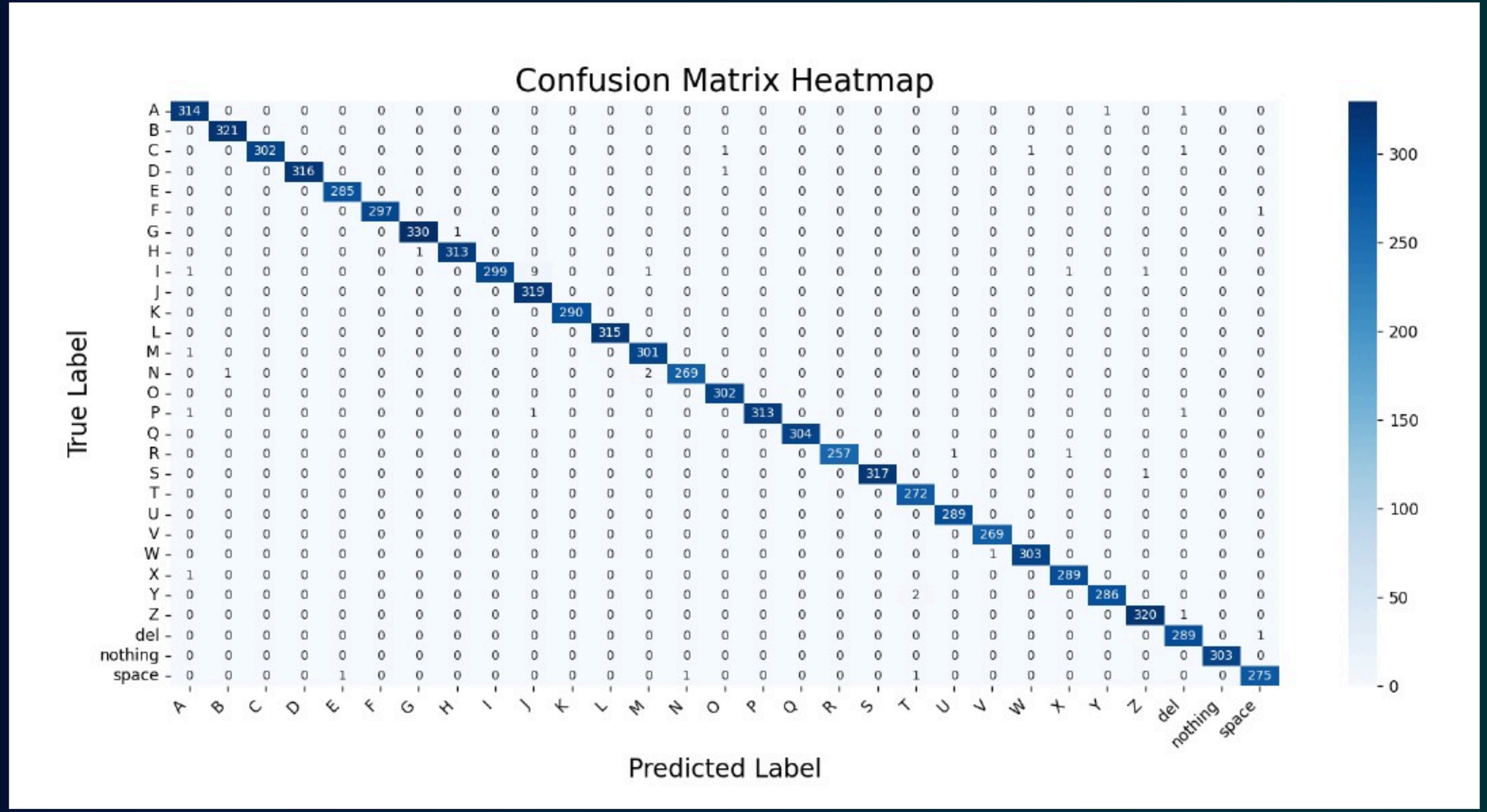
Performance Statistics



Performance Statistics



Performance Statistics



Performance Statistics

Class	Precision	Recall	F1-Score	Support
A	0.99	0.99	0.99	316
B	1.00	1.00	1.00	321
C	1.00	0.99	1.00	305
D	1.00	1.00	1.00	317
E	1.00	1.00	1.00	285
F	1.00	1.00	1.00	298
G	1.00	1.00	1.00	331
H	1.00	1.00	1.00	314
I	1.00	0.96	0.98	312
J	0.97	1.00	0.98	319
K	1.00	1.00	1.00	290
L	1.00	1.00	1.00	315
M	0.99	1.00	0.99	302

Class	Precision	Recall	F1-Score	Support
N	1.00	0.99	0.99	272
O	0.99	1.00	1.00	302
P	1.00	0.99	1.00	316
Q	1.00	1.00	1.00	304
R	1.00	0.99	1.00	259
S	1.00	1.00	1.00	318
T	0.99	1.00	0.99	272
U	1.00	1.00	1.00	289
V	1.00	1.00	1.00	269
W	1.00	1.00	1.00	304
X	0.99	1.00	0.99	290
Y	1.00	0.99	0.99	288
Z	0.99	1.00	1.00	321

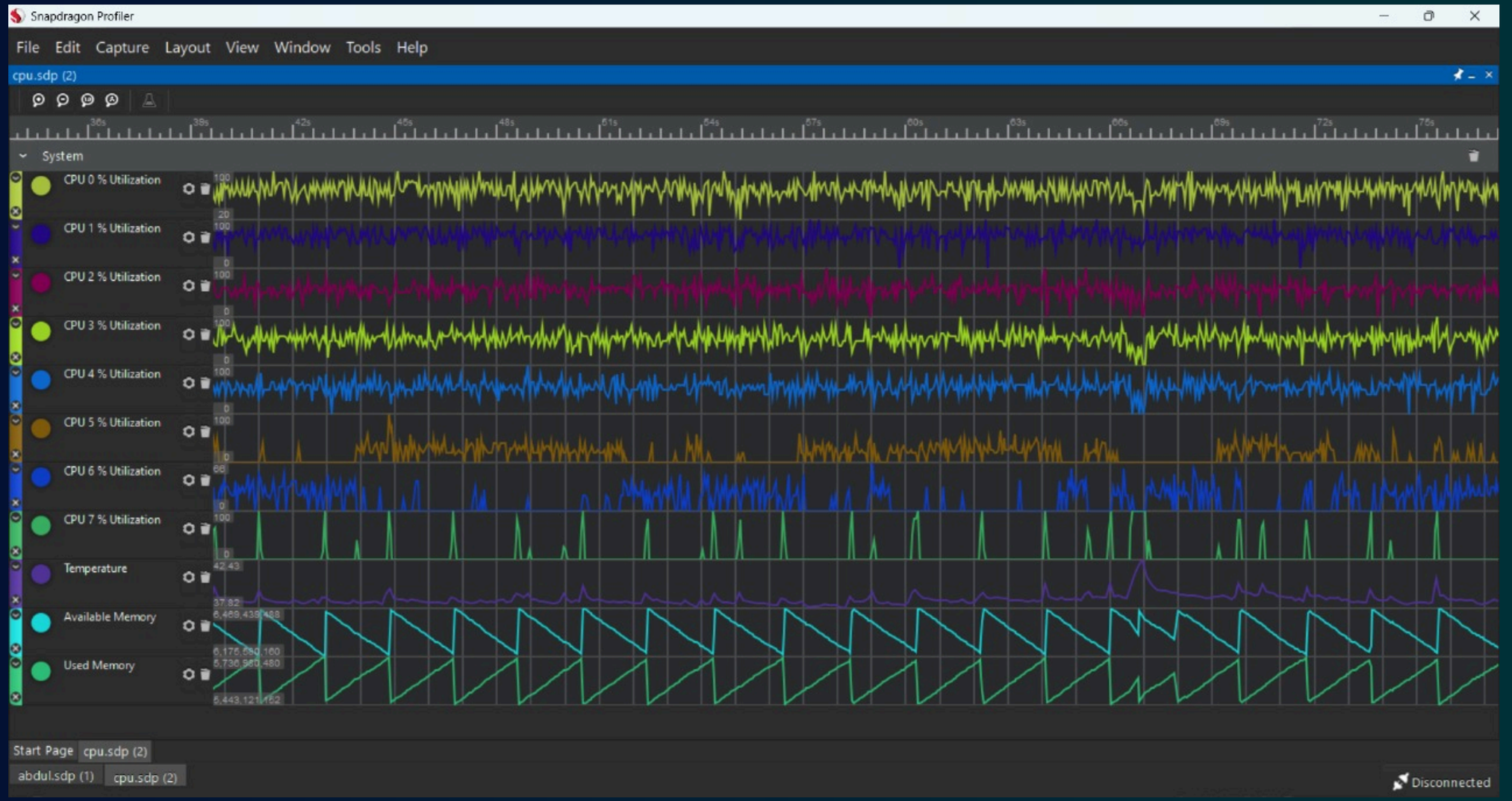
Accuracy		1.00		8700
Macro Avg	1.00	1.00	1.00	8700
Weighted Avg	1.00	1.00	1.00	8700

Benchmarking Data

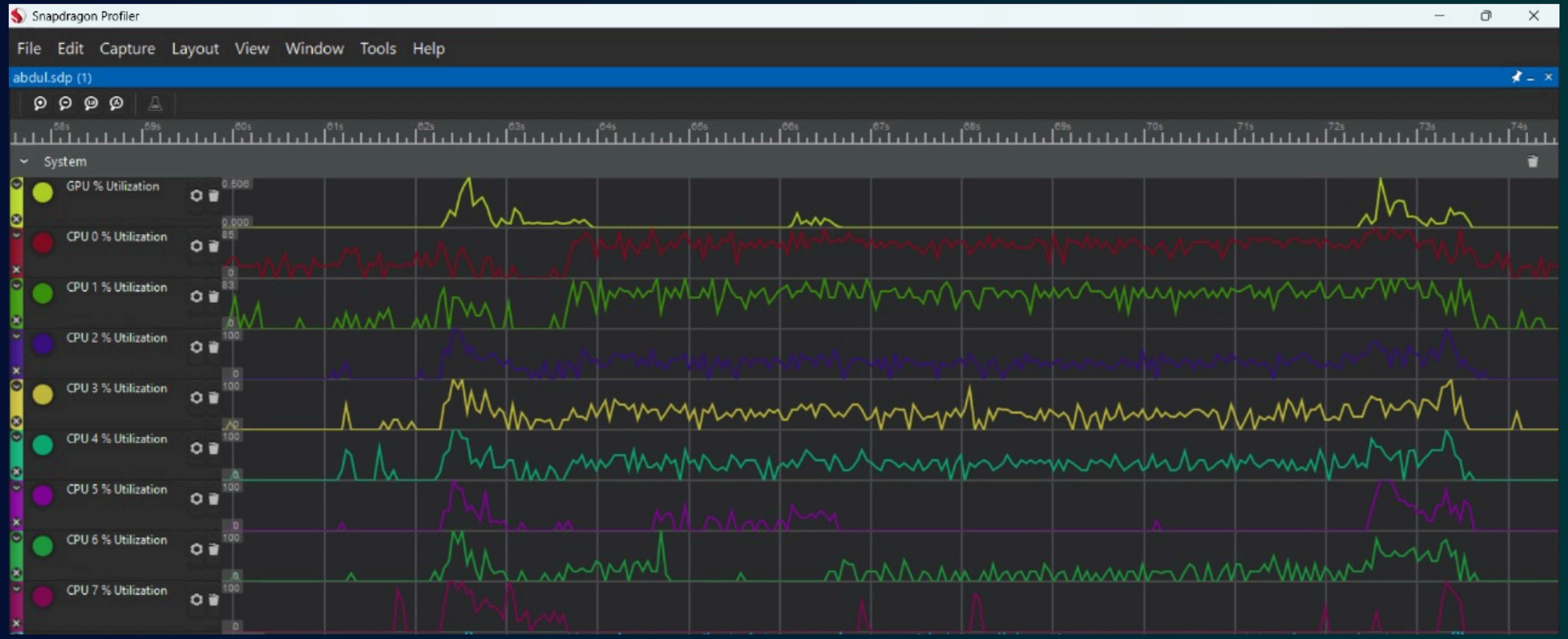
- We ran the CPU and GPU apps we developed on 6 different phones, each with a different processor. Here are our summarised findings about the inference times for each of them:

Phone (Processor)	CPU Latency	GPU Latency
Qualcomm QIDK (Snapdragon 8)	20–25 ms	18–22 ms
Samsung A35 5G (Exynos 1380)	25–29 ms	24–27 ms
Samsung M33 (Exynos 1280)	26–29 ms	24–26 ms
Nothing Phone 3a Pro (Snapdragon 7s Gen 3)	23–27 ms	22–25 ms
Honor X9B (Snapdragon 6 Gen 1)	27–29 ms	24–27 ms
Nothing Phone 2 (Snapdragon 8+ Gen 1)	22–27 ms	20–23 ms

Snapdragon Profiler Trace (CPU)



Snapdragon Profiler Trace (GPU)




Thank You