Gesture Based Text Editor on Edge

Embedded Systems Workshop Project Report

Team: Alt + F4

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

December 2, 2025

Contents

1 Problem Statement and Motivation

1.1 Problem Statement

The objective of this project was to develop a gesture-recognition smart interface on the Qualcomm Innovators Development Kit (QIDK). The primary goal was to leverage AI-on-edge features (Qualcomm AI stack) to optimize performance, creating a low-latency application capable of detecting signed letters and symbols to accurately display text in real-time.

1.2 Motivation

Gesture-based text editors act as a crucial bridge in mixed environments, particularly for individuals who are hard of hearing. While full sign language is complex and difficult for the general population to master, learning a small set of static signs (alphabets and control gestures) is significantly easier. This project aims to facilitate communication by translating these static gestures into text instantly using edge computing, ensuring privacy and speed without reliance on cloud processing.

2 Android Application Implementation

The final application was developed using Android Studio, utilizing **CameraX** for image capture and **MediaPipe** for on-device inference. The application is architected to run efficiently on both CPU and GPU delegates.

2.1 Core Logic: GestureRecognizerHelper.kt

The `GestureRecognizerHelper` class is the central component responsible for managing the MediaPipe inference pipeline. It handles the initialization of the model and processes input frames from the camera.

2.1.1 Initialization and Delegate Switching

The application supports dynamic switching between CPU and GPU delegates. This is handled in the `setupGestureRecognizer` function, which configures the `BaseOptions` builder based on the selected delegate.

```
val baseOptionBuilder = BaseOptions.builder()

when (currentDelegate) {
    DELEGATE_CPU -> {
        baseOptionBuilder.setDelegate(Delegate.CPU)
    }
    DELEGATE_GPU -> {
        baseOptionBuilder.setDelegate(Delegate.GPU)
    }
}

baseOptionBuilder.setModelAssetPath(MP_RECOGNIZER_TASK)
```

Listing 1: Delegate Configuration in setupGestureRecognizer

2.1.2 Live Stream Processing

For real-time detection, the `recognizeLiveStream` function processes `ImageProxy` objects from CameraX. It converts the image buffer to a Bitmap, applies necessary rotation, and sends the frame to MediaPipe asynchronously.

2.2 Robustness: Autocorrection Logic

To ensure the output text is coherent, we implemented a custom autocorrection feature in `GestureAutocorrector.kt`.

2.2.1 Dictionary Loading

The class loads a frequency dictionary from a CSV file (`unigram_freq.csv`) located in the assets folder. This allows the app to rank suggestions based on common usage.

2.2.2 Levenshtein Distance Algorithm

We implemented the Levenshtein distance algorithm to calculate the "edit distance" between the recognized gesture string and valid words in the dictionary. This helps identify the intended word even if a gesture was slightly misclassified.

```
private fun levenshteinDistance(s1: String, s2: String): Int {
    val dp = Array(s1.length + 1) { IntArray(s2.length + 1) }
    // ... matrix initialization ...
    for (i in 1..s1.length) {
        for (j in 1..s2.length) {
            val cost = if (s1[i - 1] == s2[j - 1]) 0 else 1
            dp[i][j] = minOf(
                dp[i - 1][j] + 1,      // deletion
                dp[i][j - 1] + 1,      // insertion
                dp[i - 1][j - 1] + cost // substitution
            )
        }
    }
    return dp[s1.length][s2.length]
}
```

Listing 2: Levenshtein Distance Implementation

2.3 UI Features and Feedback

Based on the project presentation, the app includes several user experience features:

- **Stability Buffer:** A character is only confirmed if the same gesture is detected for 15 continuous frames.
- **Visual Feedback:** The camera flash is triggered upon successful character confirmation to signal the user.
- **Control Gestures:**
  - **Space:** Triggers the ‘correctWord‘ function.
  - **Two Spaces:** Inserts a full stop.
  - **Three Spaces:** Clears the text display.

3 Model Methodology

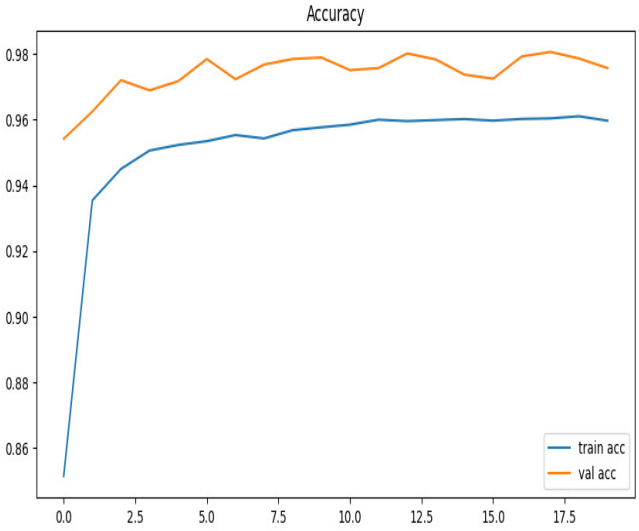
3.1 Dataset

The model was trained on a custom dataset designed for robustness:

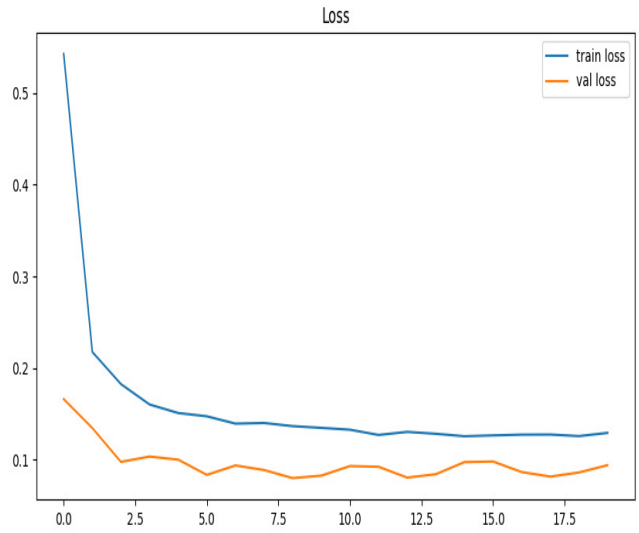
- **Classes:** 29 Total Classes (26 Alphabets, *nothing*, *space*, *delete*).
- **Volume:** 3,000 images per class.
- **Variability:** Includes variations in lighting, hand distance, and signers.

3.2 Training Performance

The model achieved high accuracy during training, with a final validation accuracy of 0.9952 and a loss rate of 0.18.



(a) Training and Validation Accuracy



(b) Training and Validation Loss

Figure 1: Training Metrics over Epochs

4 Performance Statistics

4.1 Classification Report

The model demonstrates exceptional precision and recall across all classes, as detailed below.

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
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
del	0.99	1.00	0.99	290
nothing	1.00	1.00	1.00	303
space	0.99	0.99	0.99	278
Accuracy		1.00		8700
Macro Avg	1.00	1.00	1.00	8700
Weighted Avg	1.00	1.00	1.00	8700

Table 1: Classification Report for ASL Gesture Recognition Model

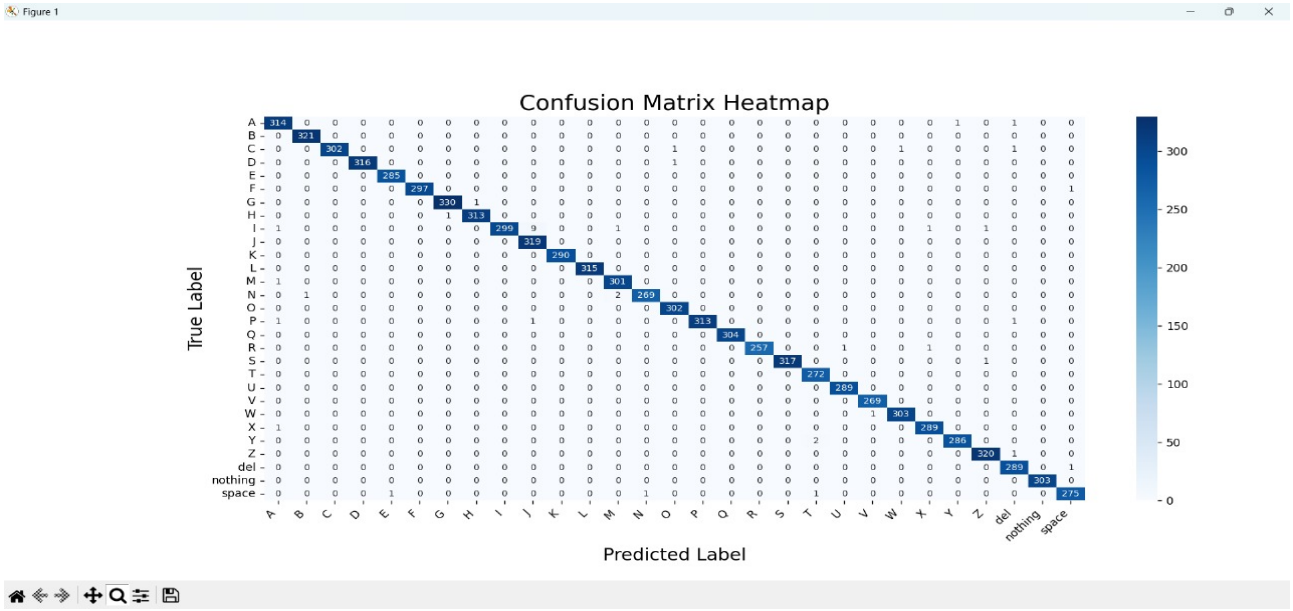


Figure 2: Confusion Matrix Heatmap

5 Hardware Benchmarking

We benchmarked the application on six different devices to compare CPU and GPU inference latency. The QIDK (Snapdragon 8) showed superior performance, with GPU latency as low as 18ms.

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

Table 2: Comparison of CPU/GPU Latency Across Devices

5.1 Resource Profiling (QIDK)

Using the Snapdragon Profiler, we analyzed the resource utilization. The traces below confirm efficient workload distribution.

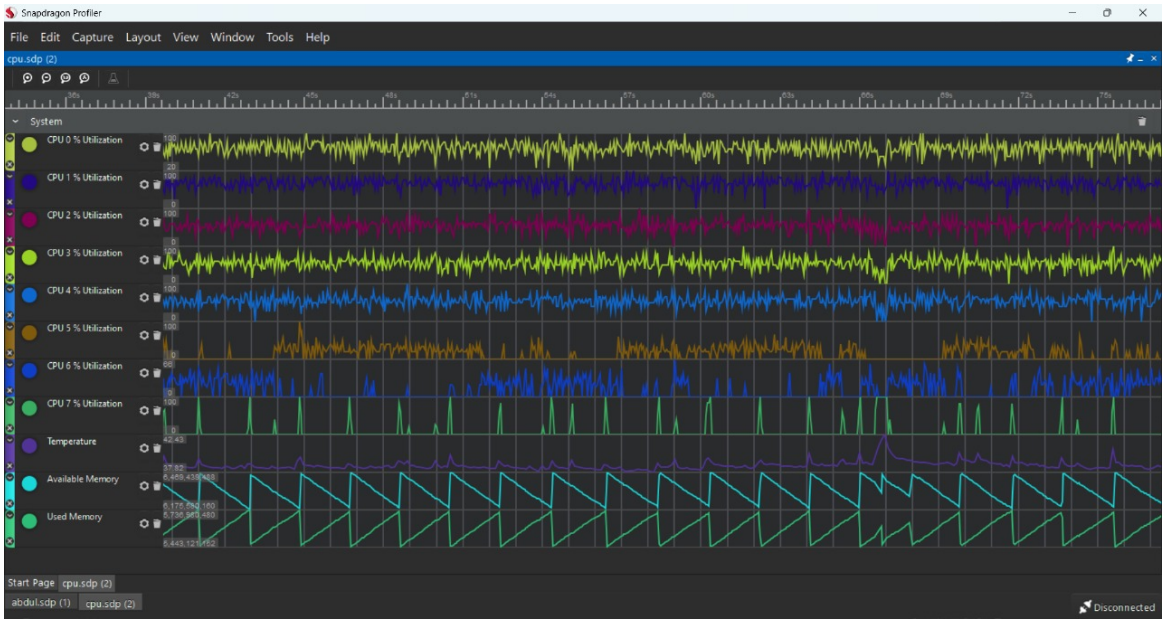


Figure 3: CPU Utilization Trace on QIDK

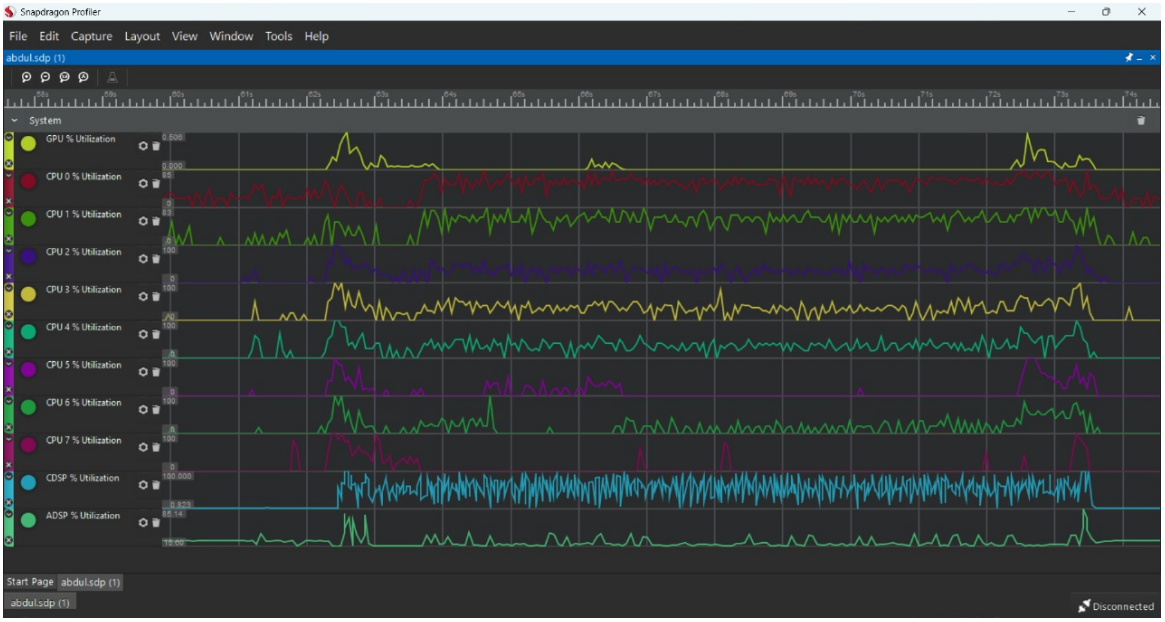


Figure 4: GPU Utilization Trace on QIDK

6 Conclusion

The project successfully delivered a robust, gesture-based text editor tailored for edge devices. By implementing custom logic for stability buffering and Levenshtein-based autocorrection directly in Android, we ensured a smooth user experience. Benchmarking results highlight that Snapdragon-based devices, particularly the QIDK, provide the low latency required for seamless real-time sign language translation.