UNIVERSITY OF PISA

Department of Information Engineering

Business and Project Management Course

SignLearnAI

Enhancing Sign Language Learning Through AI

Author:
Ivan Brillo

ACADEMIC YEAR 2024/2025

Contents

1 Inclusivity in Education: Challenges and SignLearnAI's Role

1.1 Introduction and Problem Statement

American Sign Language (ASL) is a complete, natural language that serves as the primary means of communication for many deaf and hard-of-hearing individuals in North America (1.1). Despite its importance, resources for learning ASL remain limited, particularly interactive tools that provide immediate feedback. Traditional learning methods often involve:

- In-person classes that can be limited by course schedules and cost constraints
- Mobile applications that typically offer only passive learning through flashcards and video demonstrations

With the rise of AI, new learning possibilities have emerged, attempting to avoid common drawbacks of traditional learning methods. Two approaches can be followed:

1. Contact-based approach, which involves the use of data gloves to collect finger positioning
2. Vision-based approach, which instead uses a camera to collect images. This method is typically preferred for its simplicity of use

1.2 Social Context and Educational Impact

The latest report from the World Health Organization (WHO) revealed that 466 million people were suffering from hearing loss in 2019, which amounts to 5% of the world's population. The WHO also estimated that this number would double (i.e., 900 million people) by 2050 [1].

1.2.1 Barriers to School Integration

Deaf and hard-of-hearing students often encounter significant problems in mainstream classroom environments:

- Lack of fluent sign language staff
- Social isolation due to communication barriers with peers
- Lower academic performance due to limited classroom participation

1.2.2 Promoting Inclusivity Through Technology

SignLearnAI, our open-source ASL learning application, not only supports deaf and hard-of-hearing students by providing a user-friendly platform to practice ASL but also serves as a bridge for communication with their hearing classmates. By making ASL learning accessible and engaging, the application can be adopted in inclusive classrooms to:

- Encourage hearing students to learn basic finger-spelling and sign recognition
- Enable smoother interaction between deaf and hearing students or teachers through shared language tools

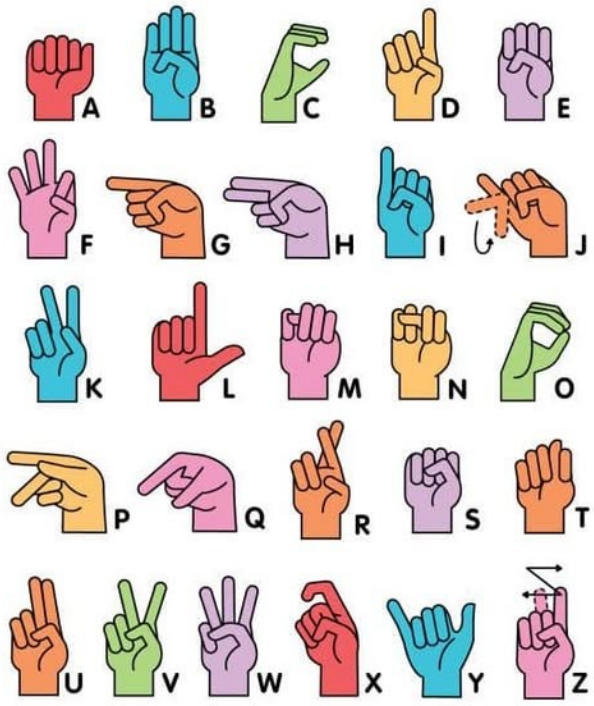


Figure 1.1: American Sign Language Alphabet

- Normalize the use of sign language as a valid and respected form of communication

1.3 SignLearnAI Approach

SignLearnAI addresses these challenges by creating an interactive AI-powered learning environment that provides:

1. Multi-format quizzes (video, text, phrase) to enhance ASL fluency in various contexts
2. Real-time recognition and feedback on ASL finger-spelling gestures
3. Adaptive learning algorithms that focus on problematic letters based on error patterns
4. Statistics visualization to guide the learning process

1.3.1 Differentiation from Existing Solutions

While several applications exist for ASL learning, SignLearnAI offers significant advantages that distinguish it from current market offerings. The comparison is performed against *Pocket Sign*, which is one of the most used apps for ASL learning (more than a million downloads on Play Store), *SignWise*, which is an online application with camera detection, and *Sign AI*, the new Nvidia platform for sign learning, released in 2025.
Most existing applications focus on video demonstrations or static images without the capability to evaluate user performance. The few that offer recognition capabilities typically:

- Do not highlight what errors users are making, but offer only an accuracy score on the sign
- Lack systematic error tracking to guide improvement

Feature	**SignLearnAI**	**Pocket Sign**	**SignWise**	**Sign AI**
AI-powered recognition	✓	×	✓	✓
Real-time feedback	✓	×	Partial	Partial
Adaptive difficulty	✓	Partial	Partial	×
Error analytics	✓	✓	×	×
Phrase practice	✓	✓	×	×
Dynamic Signs	×	×	×	✓
Open-source	✓	×	×	×

Table 1.1: Comparison of SignLearnAI with existing sign language learning applications

1.4 Future Work

While SignLearnAI provides a functional platform for ASL learning, several enhancements are planned:

- Integration of dynamic gesture recognition for letters J and Z (which require movements), as well as common ASL words
- Mobile platform adaptation for iOS and Android to make the learning phase smoother

The code for the project can be found at the following GitHub page: SignLearnAI

2 Model Training

2.1 Dataset Description

The dataset used to train the ML application is the ASL Sign Language Alphabet Pictures, which consists of 8442 images of size 1920x1920, showing different people performing ASL in various environments. In order to achieve better results, we augmented the dataset:

1. Rotated the image for each angle $\in [-15, +15]$
2. Adjusted brightness by creating a darker and a brighter image
3. Applied Gaussian blur to the image
4. Zoomed in and padded the image's borders

The resulting dataset is composed of 57607 images.

2.1.1 MediaPipe

MediaPipe is a Google open-source project that provides a variety of libraries for performing ML tasks. In particular, we used the MediaPipe Hand Landmarker, which allows detection of hand landmarks in an image with high precision and low latency (2.1).

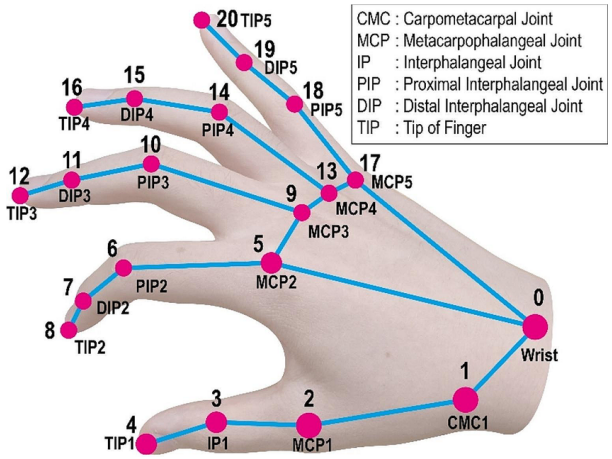


Figure 2.1: MediaPipe hand markers

The augmented dataset is analyzed via MediaPipe to obtain a dataframe with 21 points (corresponding to different MediaPipe markers), which results in 63 numerical features, since we consider the x, y, and z coordinates and the instance label (the letter performed during the image capture).

2.1.2 MediaPipe Alternative Pipeline

Instead of using MediaPipe, some recent papers ([2]) feed the whole image to a CNN in order to predict the sign (2.2). Obviously, when we deal with constrained devices such as personal computers, it is better to use pre-trained and optimized networks for the inference phase. This way, we also save a lot of training computation and keep our final model compact and efficient.

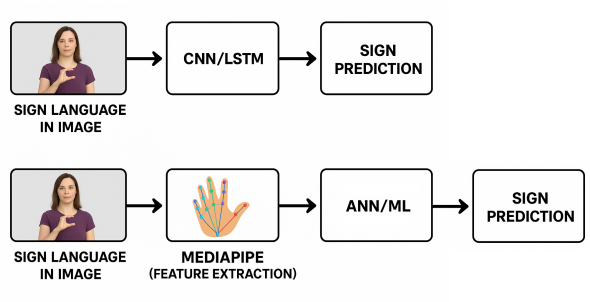


Figure 2.2: Different pipelines for sign language recognition

2.1.3 Addressing Dataset Imbalance

The dataset was split with 75% for training and the remaining 25% for testing. Unfortunately, the dataset is imbalanced (2.3), meaning that some letters have more images associated with them. Since this imbalance could compromise the model's evaluation performance, we applied ADASYN on the training set (not on the test set, otherwise the result metrics would be compromised). This method balances the different classes by generating new instances of minority classes that are coherent with other samples of the same class (2.4).

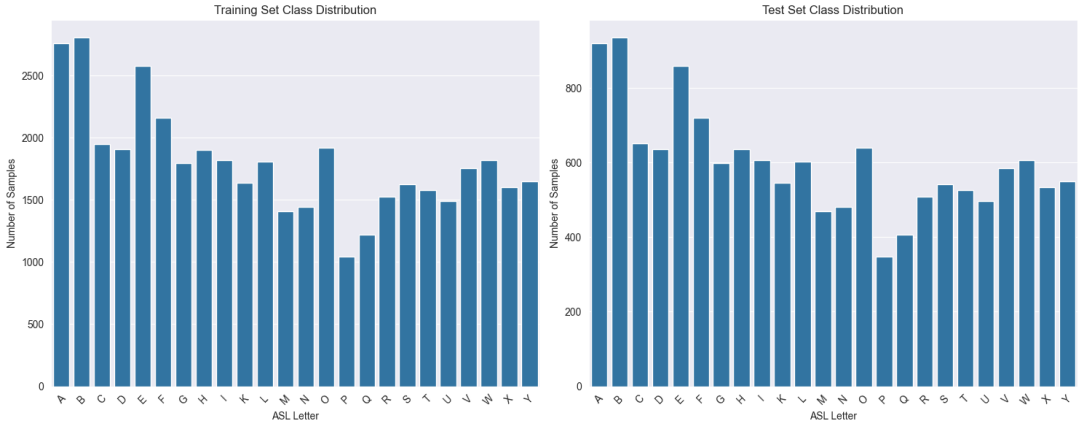


Figure 2.3: Dataset splitting and corresponding imbalance

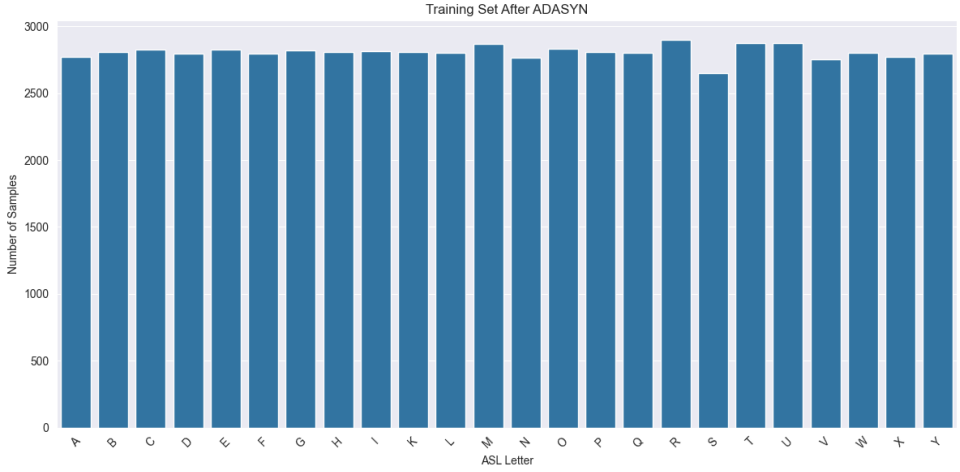


Figure 2.4: ADASYN result on training dataset

2.2 Model Selection

We trained the final multi-class classifier using different architectures: a dense neural network, a random forest, and a kNN. Then we evaluated their performance on the test set.

```
    n_features = 63
    n_classes = 24

    ann = Sequential([
        layers.InputLayer(shape=(n_features,)),
        layers.Dense(n_features), layers.Dropout(0.2),
        layers.Dense(128, activation='relu'),
        layers.Dense(256, activation='relu'),
        layers.Dense(512, activation='relu'),
        layers.Dense(512, activation='relu'), layers.Dropout(0.2),
        layers.Dense(256, activation='relu'),
        layers.Dense(128, activation='relu'), layers.Dropout(0.2),
        layers.Dense(64, activation='relu'),
        layers.Dense(n_classes, activation='softmax')
    ])

    rf = RandomForestClassifier(100, random_state=42, n_jobs=-1)
    knn = KNeighborsClassifier(n_neighbors=8, n_jobs=-1)

```

As we can see, all the models are highly accurate. Since our application should achieve low latency to ensure smooth user interaction, we chose the random forest model, that ensures the lowest evaluation time per sample. The results are displayed in the following images (2.5, 2.6).

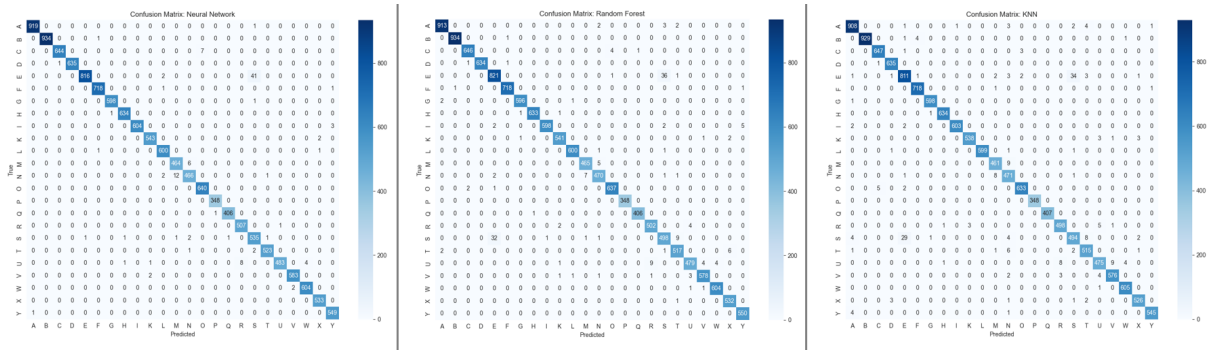


Figure 2.5: Confusion Matrix for the tested models

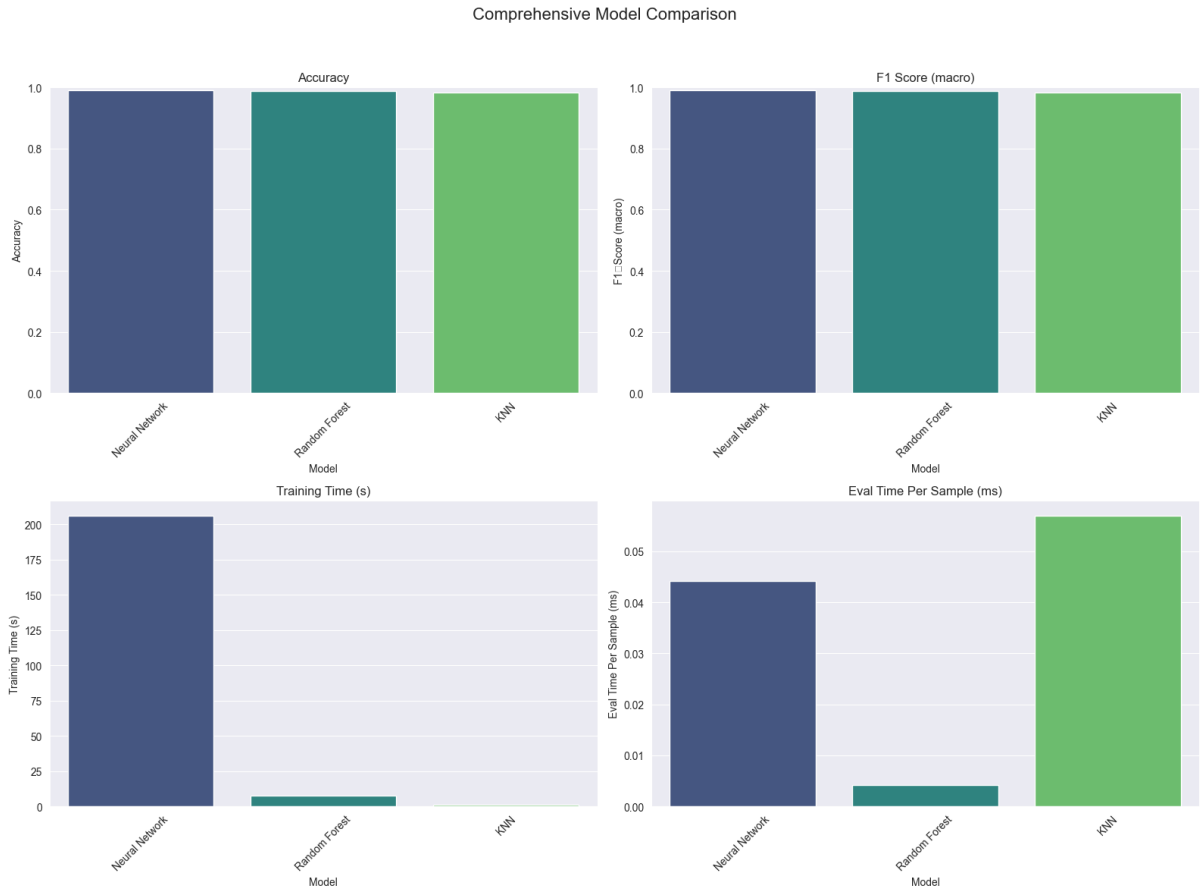


Figure 2.6: Statistics for the tested models

3 Application Development

The Quiz application has been built using a specific Python library called *customTkinter*. From the home screen (3.1), we can access the three main parts of the application: individual letters quiz, phrase practice, and the statistics pop-up.

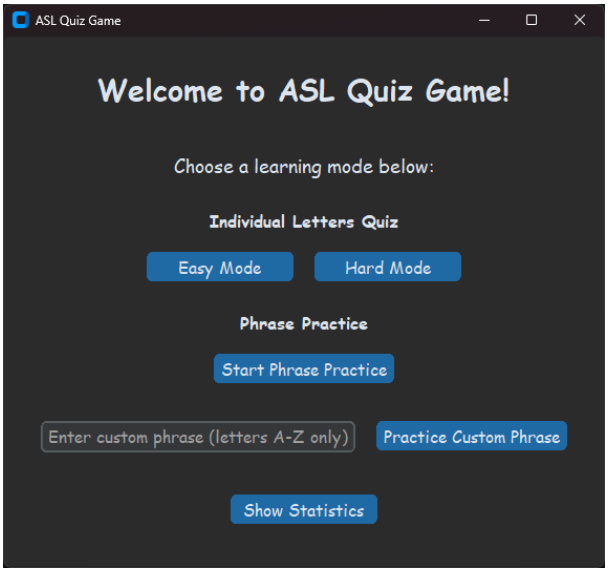


Figure 3.1: Home Screen of the Application

3.1 Individual Letters Quiz Screen

In this section, two modalities are present: one for beginners and another for expert users. The easy mode allows users to actually see an image of the sign that needs to be performed, while hard mode requires the user to remember exactly how to mimic the sign (3.2). Each of the two modalities provides a mixed quiz that combines actual quiz performance by the user with quiz images for sign recognition (3.3). The different modalities and the letters to perform are chosen using a scoring system that takes into consideration previously made errors. The details of the implementation are deferred to the next section.

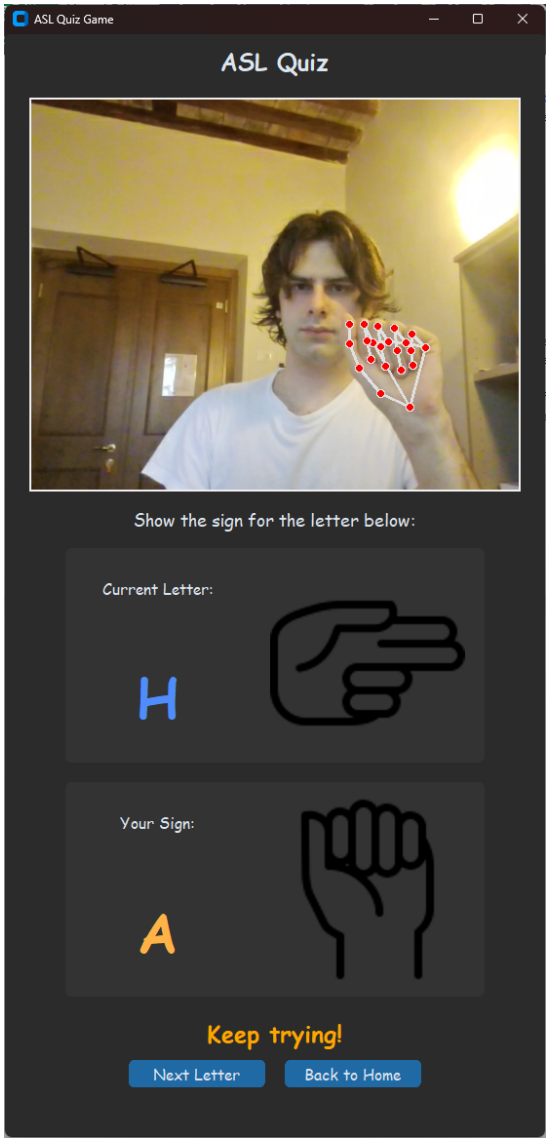
3.1.1 Weight Scoring

In this section, we present how the application chooses between text-video quizzes and which letter to select each time. To avoid zero divisions, every error entry is initialized at one.

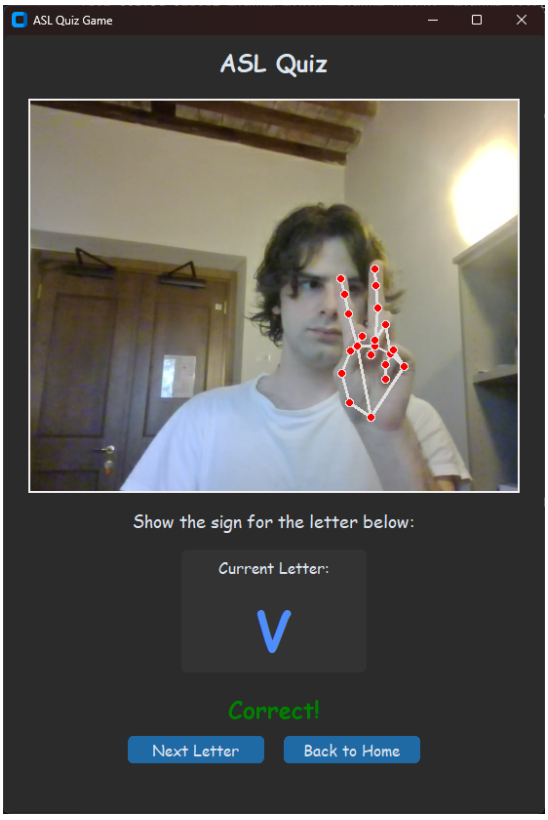
Weight scoring for text-video decisions

We want to provide the right balance between text and video quizzes, while favoring the mode with the most mistakes. With probability $\epsilon = 0.3$, we choose randomly between text or video quiz to prevent overfitting to one mode and guarantee some degree of randomness. Now we define some variables:

- E_v and E_t: Number of errors for video and text
- N_v and N_t: Number of evaluations for each mode



(a) Easy Mode Screenshot



(b) Hard Mode Screenshot

Figure 3.2: Comparison of Easy and Hard Modes

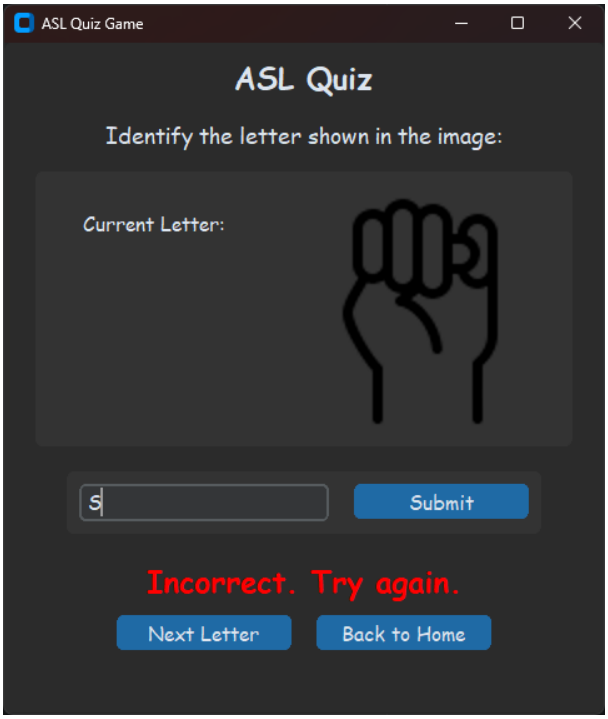


Figure 3.3: Text Quiz for both hard and easy modes

We proceed by calculating the error rate for each mode:

$$r_v = \frac{E_v}{N_v}, \qquad r_t = \frac{E_t}{N_t} \tag{3.1}$$

Then, we compute the balancing terms, ensuring that modes with fewer evaluations aren't unfairly penalized. The square root prevents extreme weighting from imbalanced test counts.

$$\text{bal}_v = \sqrt{\frac{N_t}{N_v}}, \qquad \text{bal}_t = \sqrt{\frac{N_v}{N_t}} \tag{3.2}$$

We adjust the scores for video and text quizzes by multiplying the error rate by the corresponding balancing term.

$$\text{score}_v = r_v \cdot \text{bal}_v, \qquad \text{score}_t = r_t \cdot \text{bal}_t \tag{3.3}$$

Finally, we calculate the probability of doing a video quiz and choose accordingly.

$$P(\text{video}) = \frac{\text{score}_v}{\text{score}_v + \text{score}_t} \tag{3.4}$$

Weight scoring for next letter decisions

We want to provide the right balance between letters, while favoring the most mistaken ones. With probability $\epsilon = 0.3$, we choose randomly. If we do not choose at random, we define $w_l = \text{errors}_l^{(t)}$ as the number of errors for letter l in mode t (video/text). We assign the probability of choosing a letter in the following way:

$$P(l) = \frac{w_l}{\sum_{j \in \text{letters}} w_j} \tag{3.5}$$

How errors are counted during the quiz

The error for text mode is calculated considering the number of attempts n made to get the right answer (or adding 1 if the letter is skipped via the skip button):

$$\text{errors\_text} = \text{errors\_text} + \frac{n}{n+1}$$

The same is added to the per-letter count. Meanwhile, for the video and the corresponding letter, the following term is added, considering the number of seconds t the player takes to mimic the right sign (add 1 if they are unable and skip the letter):

$$t = \min\left(\frac{\Delta t}{10}, 1\right) \tag{3.6}$$

For the first 2 seconds, we assign zero error. After that amount of time, we add the t term to the per-letter error counting as well.

3.2 Phrase Practice Screen

In this section, we can choose to practice predefined phrases or custom ones. Users in this section can gain confidence in performing entire phrases at a sustained rate. An image of the screen is shown below (3.4).

3.3 Statistics Screen

The last screen of the application is the statistics screen. It provides useful information about the letters that are being mistaken most during the entire learning session. By simply looking at these statistics, users can gain interesting and useful insights into their learning path (3.5).

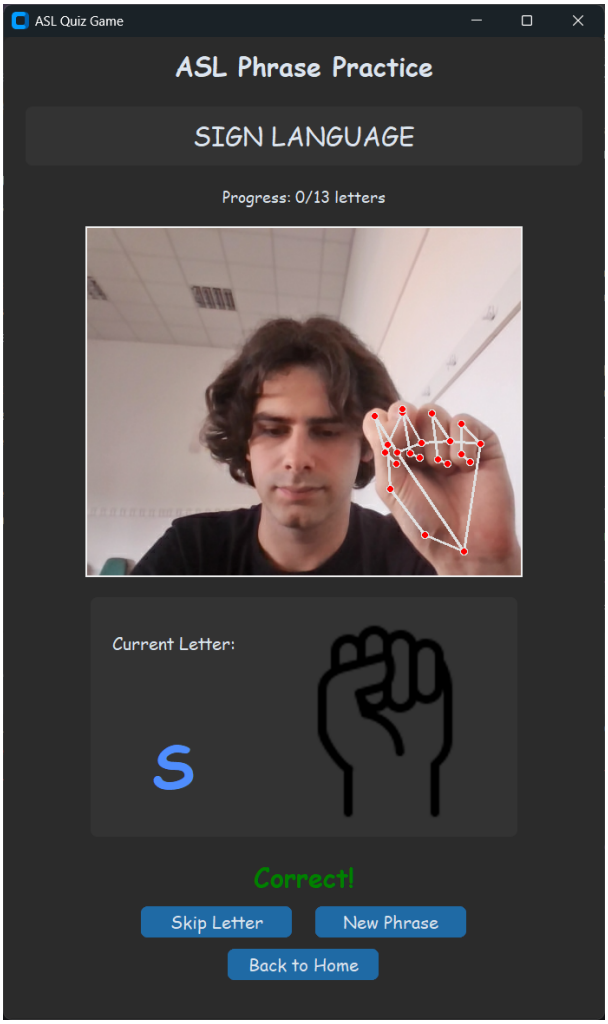


Figure 3.4: Phrase Practice: ”SIGN LANGUAGE”

Figure 3.5: Statistics after one minute of learning

Bibliography

[1] Adeyanju, I.A., Bello, O.O., & Adegboye, M.A. (2021). Machine learning methods for sign language recognition: A critical review and analysis. *Intelligent Systems with Applications*, *12*, 200056. https://doi.org/10.1016/j.iswa.2021.200056

[2] Alsharif, B., Alalwany, E., Ibrahim, A., Mahgoub, I., & Ilyas, M. (2025). Real-Time American Sign Language Interpretation Using Deep Learning and Keypoint Tracking. *Sensors*, *25*(7), 2138. https://www.mdpi.com/1424-8220/25/7/2138