Sign Language Recognition for Communication

Kwok Keith, Gizelle Lim Yin Xuan, Jun Kiat Lim

Singapore University of Technology and Design

20 April 2025

`https://github.com/kwokkeith/ASL-Translation-Model.git`

Abstract

This project presents a real-time American Sign Language (ASL) recognition system capable of classifying both static and dynamic gestures. A binary logistic regression model first distinguishes between static and dynamic gestures using hand keypoint movement. Based on the classification, the input is routed to either a Feed-Forward Neural Network (for static gestures, i.e., alphabets and digits) or an LSTM-based model (for dynamic gestures, i.e., *"hello"*, *"my"*, *"name"*, *"your"*, *"what"*). The system uses MediaPipe for keypoint extraction and applies data pre-processing techniques including PCA and temporal augmentation. The combined model achieves strong generalisation on out-of-domain data. The output is finally passed to a LLM to produce structured sentences demonstrating practical use as an ASL communication tools.

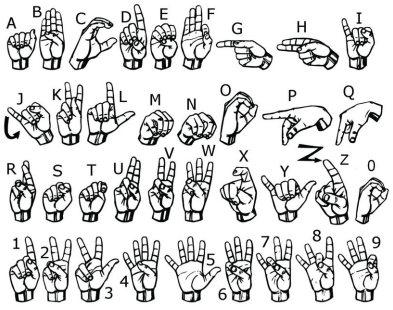
Keywords: Area Under Curve (AUC), American Sign Language (ASL), Long Short-Term Memory (LSTM), Principal Component Analysis (PCA), Stochastic Gradient Descent (SGD), ASL, Feedforward Neural Network (FNN), Receiver Operating Characteristics (ROC), Large Language Model (LLM)

1 Introduction

ASL is a key mode of communication for the Deaf and Hard-of-Hearing community, comprising both static (i.e., letters and numbers) and dynamic gestures (i.e., full words or phrases). While existing models perform well on static gesture recognition (see Figure 1a), many struggle to accurately detect dynamic gestures (see Figure 1b), which limits their usefulness in real-world ASL communication.

This report proposes a model capable of recognising both static and dynamic ASL gestures by first classifying the gesture type using a binary classifier. Based on the result, the gesture is passed to either a static gesture classifier (for alphanumeric signs) or a dynamic gesture classifier (for phrase gestures like *my*, *name*, *your*, *what*, and *hello*). Finally, LLM was used to structure an english sentence output from ASL.

This three-stage approach reflects the structure of ASL and improves recognition accuracy by using specialised models for each gesture type.



(a) The 26 letters and 10 digits of ASL (Teak-Wei Chong and Boon-Giin Lee, 2018)

(b) Phrase gestures for ASL (Wonder-Print, 2025)

Figure 1: ASL gestures comprised of both static and dynamic gestures

2 Dataset Collection

2.1 Introduction

While large volumes of data were readily available for static gestures, sourcing data for dynamic gestures proved challenging. As a result, datasets for dynamic gestures were collected. Although several static gesture datasets were available online, they were not used in order to maintain consistency in data format, capture conditions, and keypoint extraction across both static and dynamic gestures.

2.2 Datatype

Keypoint extraction is performed using MediaPipe (Google, 2025), a lightweight and efficient framework for detecting human body landmarks (Lugaresi et al., 2019). Landmarks for Face, Pose, Left hand and Right hand were obtained. Each landmark point is in the shape of (x, y, z)-coordinates. By collecting the landmark points, it aims to capture only the important context, gestures of the user. This resulted in a feature size of **1,662** collated dimensions (refer to Table 1).

Landmark	Feature Size
Face	1,404
Pose	132
Left-hand	63
Right-hand	63

Table 1: Feature size composition for different landmarks

2.3 Model Training/Test Dataset

Multiple videos capturing the performance of both static and dynamic gestures were recorded. Individual frames from these videos were processed using MediaPipe to extract the corresponding keypoints.

Static: A comprehensive dataset consisting of ASL alphabets *(A-Z)* and numbers *(0-9)* was collected.

A total of **43,200** frames were collected (refer to Table 2).

Action	Number of Frames per label
A-Z, 0-9	1,200

Table 2: Summary of static action sequence data collected

Dynamic: Sign language representations of the phrases *hello, my, name, what,* and *your* were collected. Each recorded sequence consisted of 60 frames. These five phrase actions were chosen to demonstrate the concept of action recognition while keeping the dataset size manageable.

A total of **810** sequences were collected (refer to Table 3).

Action	Number of Sequences
hello	180
your	180
my	150
name	150
what	150

Table 3: Summary of action sequence data collected

The dataset size was incrementally increased to prevent model divergence during training. Additionally, more sequences were collected for *hello* and *your* due to their similarity in action sequence, providing more samples for testing.

2.4 Out-of-domain Dataset

To evaluate the static model's generalisability, an out-of-domain dataset was collected from three different participants. Table 4 presents the total number of samples collected.

Action	Number of Frames per label
A-Z, 0-9	360

Table 4: Summary of out-of-domain static action sequence data collected

Six different participants performed five sequences for each action as part of the dynamic gesture set. Table 5 illustrates the dataset distribution. This additional dataset ensures that the model is tested beyond the training/testing domain.

Action	Number of Sequences
hello	30
your	30
my	30
name	30
what	30

Table 5: Summary of out-of-domain action sequence data collected

3 Proposed Model

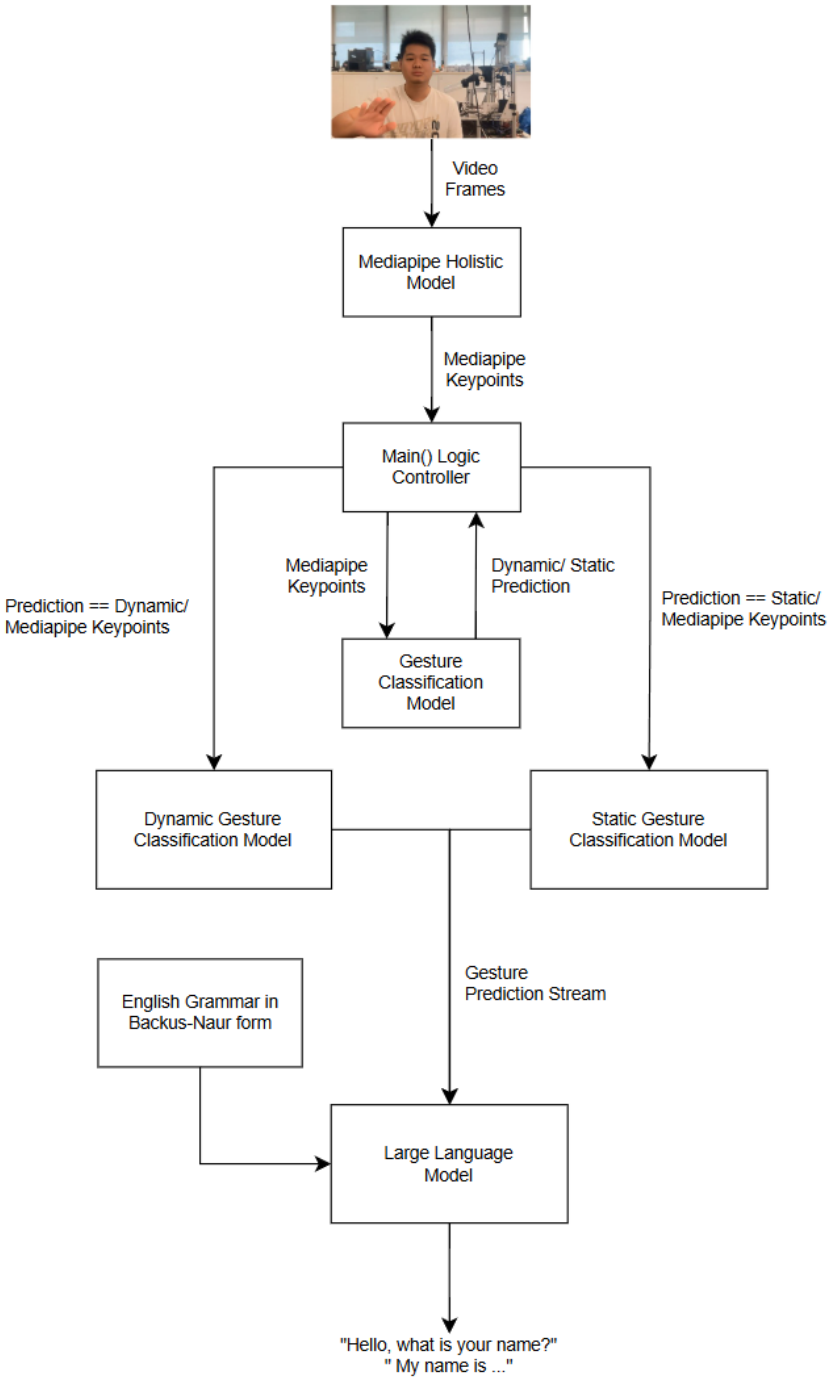


Figure 2: Proposed Model Approach for ASL Detections

The proposed model processes the last 30 frames of a live video feed to produce a gesture prediction, as illustrated in Figure 2. The approach begins with the extraction of Mediapipe Keypoints using the Mediapipe Holistic model which will be passed into a logic controller. The logic controller will first pass the keypoints into a gesture classification model that determines whether the sequence of frames corresponds to a static or dynamic gesture. If classified as static, the input is forwarded to a Static Gesture Classification model, which specialises in recognising alphanumeric gestures. If classified as dynamic, it is passed to a Dynamic Gesture Classification model, which identifies one of five phrase

gestures: *my*, *name*, *your*, *what*, and *hello*.

This approach is well-suited for ASL, which comprises both static and dynamic gestures. Static gestures are defined by a single hand shape or pose, while dynamic gestures involve movement across a sequence of frames. Employing separate specialised models for static and dynamic gesture recognition enables the system to better handle the structural differences between the two types of gestures. As a result, this hybrid approach enhances both classification accuracy and computational efficiency by reducing the complexity of each sub-task and tailoring the recognition process to the specific characteristics of ASL.

Once individual gestures are recognised by the classification model, their labels are passed sequentially to a LLM. This stream of predictions forms the input for the LLM, which uses a Backus–Naur form-defined grammar to generate coherent English sentences. For example, the gesture sequence *hello*, *what*, *your*, *name* may be translated into "Hello, what is your name?". This integration enables meaningful dialogue reconstruction from ASL input, allowing the model to infer intent and resolve ambiguities through contextual understanding.

4 Gesture Classification

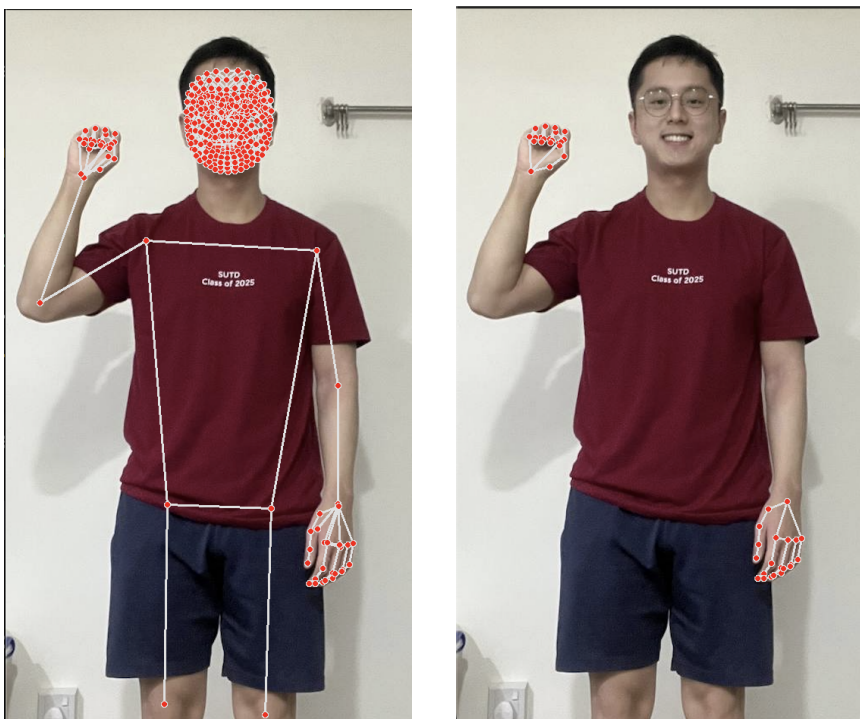
4.1 Introduction

The gesture type classifier improves the system by determining whether an input gesture is static or dynamic, allowing it to route the gesture to a specialised model accordingly. This separation leads to greater accuracy, as static and dynamic gestures require different types of feature extraction and temporal analysis.

4.2 Data Pre-processing

4.2.1 Extracting Data

Each data point in the original dataset consists of 60 frames, with each frame containing 1662 keypoints. However, within the context of real-time prediction from a live video feed, computational efficiency becomes crucial to ensure the timely processing of landmarks in response to a continuous stream of incoming frames. To accommodate this, the input size was reduced by processing only the first 30 frames, focussing on the hand keypoints, as these represent the most discriminative features for distinguishing between static and dynamic gestures. This optimisation achieves a balance between performance and responsiveness, thereby enabling the model to function effectively in real-time applications.



(a) Before processing: All keypoints (b) After Processing: Hand Keypoints

Figure 3: Visualisation of data reduction

4.2.2 Reshaping Keypoints

After the hand keypoints for a given frame are extracted, a validation step is performed to ensure that the keypoints for each individual hand conform to the expected shape of (63,). Once successfully validated, the combined keypoints are reshaped into a structured format of (2, 21, 3), representing 21 landmarks per hand, each comprising 3 coordinates: x, y, and z.

4.2.3 Depth Estimation Scaling

To account for variations in hand size and distance from the camera, the hand movement is normalised by scaling the `(x, y)` coordinates of the hand keypoints to fit within a fixed, arbitrary bounding box of size `100 × 100`. This approach ensures consistency in the representation of movement across different frames and subjects.

For each hand, a bounding box is first computed by identifying the minimum and maximum `x` and `y` values among its keypoints. A uniform scaling factor is then calculated based on the larger of the bounding box's dimensions—either width or height. The keypoints are subsequently translated and scaled to fit within a `100 × 100` space. The choice of `100` as the target dimension is arbitrary, but it provides a consistent scale for gesture analysis while preserving the spatial relationships between keypoints.

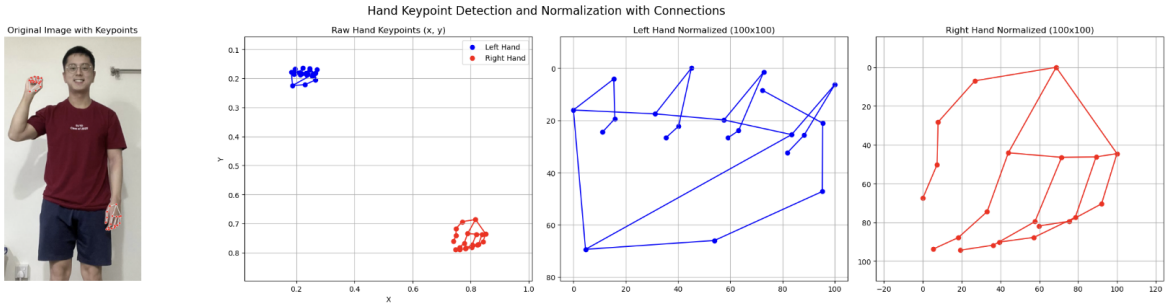


Figure 4: Hand Keypoints normalisation

4.2.4 Dimension Reduction

Although the dimensionality was reduced from `1662` features (all body, face, and hand keypoints) to `126` features (hand keypoints only), further dimensionality reduction is beneficial. To distinguish between static and dynamic gestures while improving computational efficiency, the model was trained using the average hand movement over the most recent `30` frames. This approach reduces input complexity while retaining sufficient information to effectively differentiate between gesture types.

Each average movement data point is computed by first tracking the `(x, y)` coordinates of hand keypoints over time. For each pair of consecutive frames, the Euclidean distance between corresponding keypoints is calculated to quantify the magnitude of movement. These distances are then averaged across all keypoints to produce a single movement value per frame.

As a result, a sequence of `30` scalar values, each representing the average hand movement in one frame, is generated for every gesture instance. The use of a single aggregated feature per frame also ensures the system remains lightweight and responsive for real-time deployment.

4.3 Model

4.3.1 Model Overview

Logistic regression was chosen for the binary classification of gestures (static vs. dynamic) due to its balance of simplicity, efficiency, and effectiveness. Given that the distinction between static and dynamic gestures can be captured using average hand movement, logistic regression offers a highly interpretable model that performs well on linearly separable data. It also ensures fast inference and low computational cost, which is ideal for real-time applications. Furthermore, its probabilistic output allows for flexible threshold tuning to adapt to varying sensitivity requirements.

4.3.2 Training Data Optimisation

To improve the reliability and sensitivity of gesture classification, a weighted averaging strategy was adopted to aggregate hand movement across a sequence of frames. Unlike a simple unweighted mean,

which treats all frames equally, weighted averaging allows the model to place greater emphasis on the most informative frames within a temporal window. This is particularly important in dynamic gesture recognition, where recent or high-magnitude movements often have more impact than earlier or subtler ones. By introducing weighting schemes, we enable the system to better capture temporal dependencies and motion characteristics that distinguish static from dynamic gestures, thereby enhancing classification accuracy and robustness in real-time applications. The following weighting strategies were explored:

- **Linear weighting**: Assigns gradually decreasing weights from the most recent to the earliest frame.
- **Exponential weighting**: Amplifies the influence of the most recent frames by exponentially decaying the importance of earlier frames.
- **Peak weighting**: Emphasises frames with the highest individual movement magnitudes, which is useful for gestures characterised by sharp motion bursts.

4.4 Evaluation

4.4.1 Overview

In order to identify the most effective weighting strategy, multiple evaluation metrics were employed to assess classification performance. As the logistic regression model was trained using a single feature—the average hand movement—it converged rapidly. Consequently, analysing model performance based on the number of training epochs yielded limited insight. Instead, alternative evaluation methods were prioritised, offering a more meaningful reflection of the model's effectiveness and discriminative capability.

4.4.2 ROC Curve and AUC Values

Initially, all three weighting strategies were plotted on the Receiver Operating Characteristic (ROC) curve. Subsequently, the Area Under the Curve (AUC) values were calculated for each strategy to quantify and compare their classification performance.

Figure 5: ROC curve

As summarised in Table 6, both the Linear and Exponential weighting strategies achieved the highest AUC value of `0.96`, indicating excellent classification performance. The Peak weighting strategy followed closely with an AUC of `0.94`.

Weightage Strategy	AUC Value
Linear	0.96
Exponential	0.96
Peak	0.94

Table 6: AUC values of the different Weightage Strategies

4.4.3 Youden's J Statistic

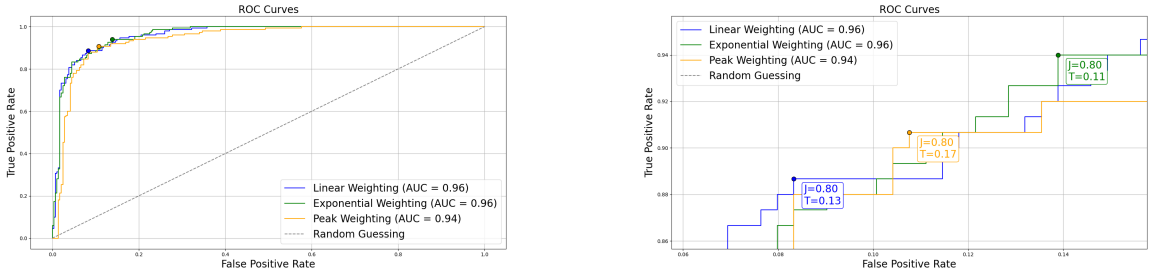
To evaluate and compare the effectiveness of each weighting strategy, it is essential to determine the optimal classification threshold for each model. This allows for a fair comparison based on each model's best possible performance. A widely accepted method for identifying this optimal threshold is by maximising Youden's J statistic (Ruopp et al., 2008), a metric that balances sensitivity and specificity. Youden's J is defined by the formula:

$$J = \text{TPR} - \text{FPR},$$

where TPR (True Positive Rate) measures the proportion of actual positives correctly identified, and FPR (False Positive Rate) represents the proportion of actual negatives incorrectly classified as positives. The value of J ranges from `-1` to `1`, with a higher value indicating better overall classification performance.

Since the Receiver Operating Characteristic (ROC) Curve plots the trade-off between TPR and FPR at various threshold levels, it provides a natural basis for computing Youden's J. By identifying the point on the ROC curve where J is maximised, the threshold that yields the best balance between correctly identifying dynamic gestures (true positives) and avoiding misclassification of static gestures (false positives) can be extracted.

By applying this method to each weighting strategy, it becomes possible to visualise and annotate the optimal thresholds on their respective ROC curves. This facilitates a more meaningful comparison between models, ensuring that each is assessed under its most favourable decision boundary.



(a) Zoomed out ROC curve

(b) Zoomed in ROC curve

Figure 6: ROC Curves

Table 7 summarises the results:

Weightage Strategy	Maximum Youden's J Value	Threshold Value
Linear	0.80	0.13
Exponential	0.80	0.17
Peak	0.80	0.11

Table 7: Table for the Maximum Youden's J Value and its corresponding Threshold Value

4.4.4 Computational Efficiency

Following the identification of optimal thresholds for each weighting strategy, an evaluation of computational efficiency was conducted. This metric is particularly important given the real-time requirements of gesture classification in live video streams, where both speed and accuracy are essential. While all weighting strategies were evaluated on an identical number of data samples, variations in total processing time were observed, reflecting underlying differences in computational complexity. These disparities are illustrated in Figure 7, which presents the time required to process `1,252` samples for each respective method.

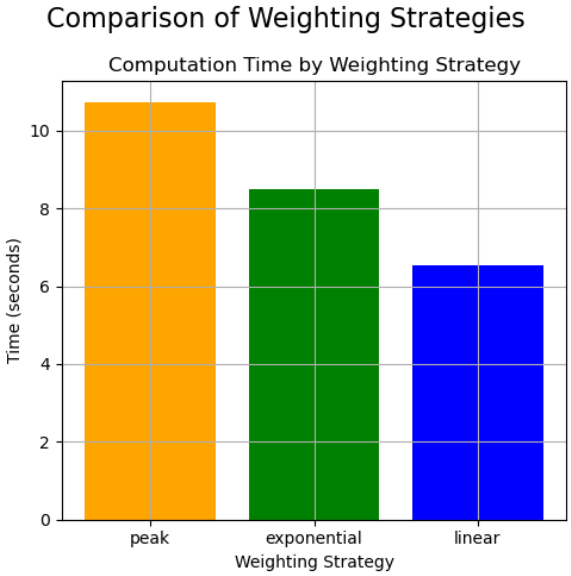


Figure 7: Time Cost of Different Weighting Strategies

4.4.5 Conclusion and Model Selection

Based on the comparative evaluation of the three weighting strategies (Linear, Exponential, and Peak), it is evident that all approaches achieved similarly high classification performance, with AUC values of `0.94` or higher and identical maximum Youden's J values of `0.80`. However, when computational efficiency is taken into consideration, the Linear weighting strategy exhibits a clear advantage, consistently requiring the least amount of processing time.

Given the requirement for real-time responsiveness in live video gesture classification systems, the Linear weighting strategy was selected for the final model. This decision is supported by its strong performance at the optimal classification threshold of `0.13`, which maximises Youden's J statistic whilst ensuring computational efficiency. Among the strategies evaluated, the Linear weighting approach offers the most effective balance between classification accuracy and runtime performance, rendering it the most suitable candidate for deployment in time-sensitive applications.

Using this optimal threshold, the resulting confusion matrix for the out-of-domain test dataset is presented below, with an associated Macro F1 score of: `0.8765`.

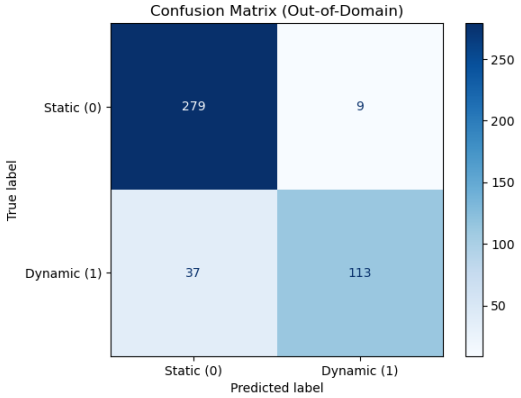


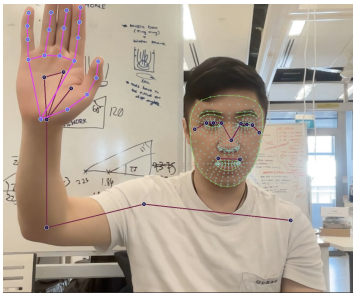
Figure 8: Confusion Matrix for Best Model

5 Action Recognition

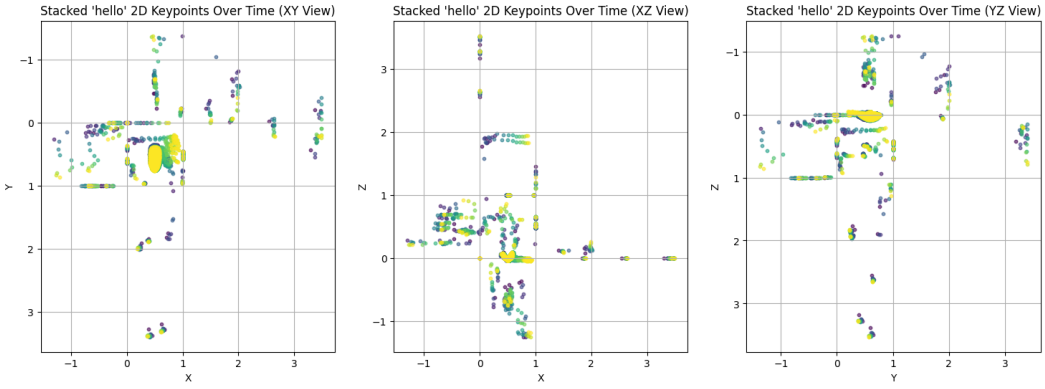
This section explores the use of LSTM neural network in predicting action sequences of ASL for phrases *hello*, *my*, *name*, *what* and *your*.

5.1 Data Exploration

Each action sequence was plotted along the x, y and z axes. The sequences plotted below are based off the first 30 frames with frame skipping of one.

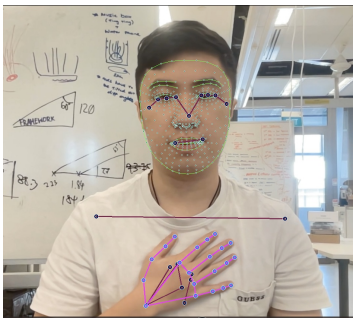


(a) Landmarks for “hello” action

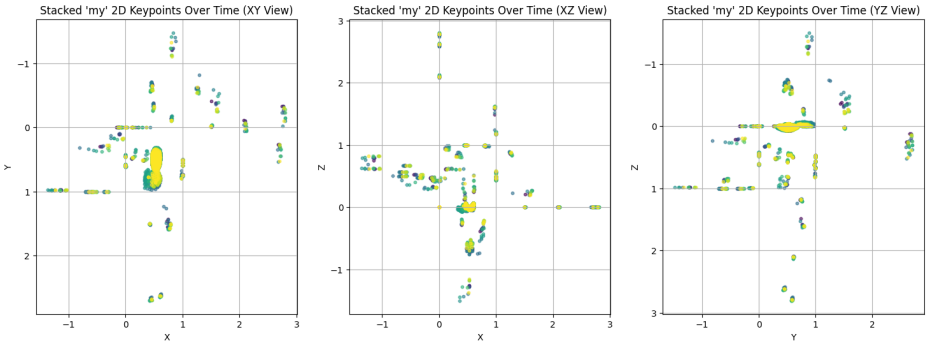


(b) Sample “hello” action sequence plotted for 30 frames with frame skipping

Figure 9: “hello” action sequence

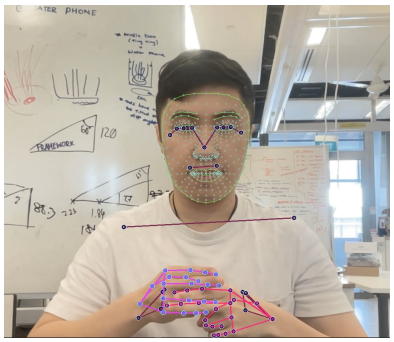


(a) Landmarks for “my” action

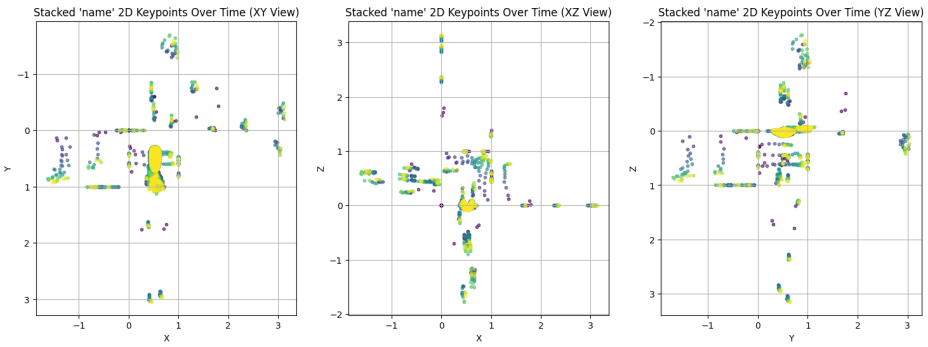


(b) Sample “my” action sequence plotted for 30 frames with frame skipping

Figure 10: “my” action sequence

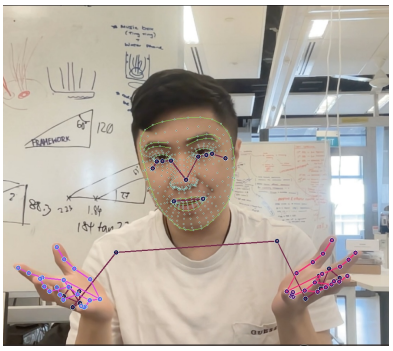


(a) Landmarks for “name” action

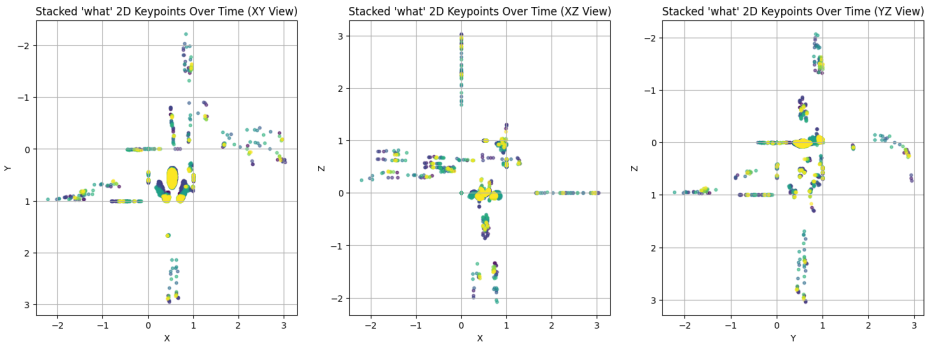


(b) Sample “name” action sequence plotted for 30 frames with frame skipping

Figure 11: “name” action sequence

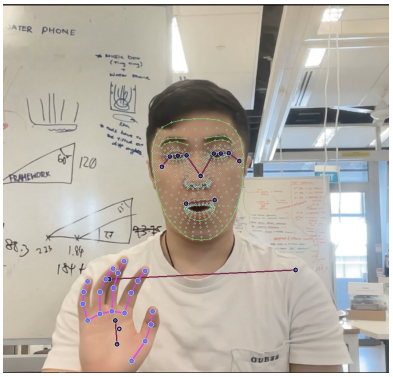


(a) Landmarks for “what” action

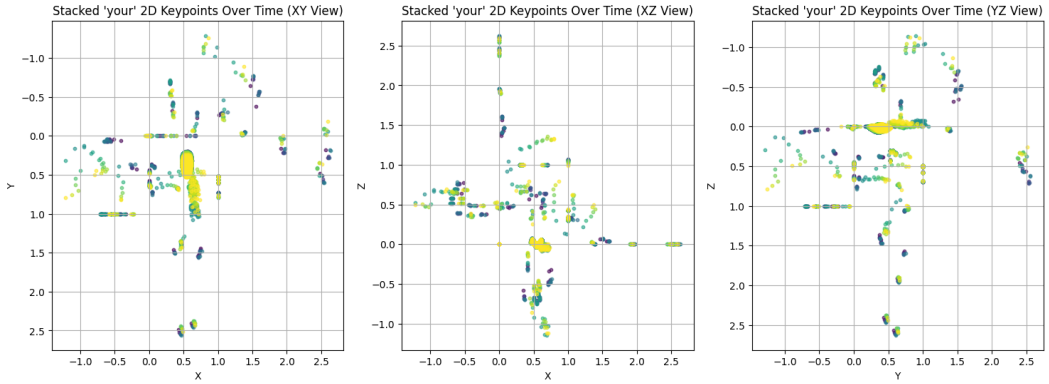


(b) Sample “what” action sequence plotted for 30 frames with frame skipping

Figure 12: “what” action sequence



(a) Landmarks for “your” action



(b) Sample “your” action sequence plotted for 30 frames with frame skipping

Figure 13: “your” action sequence

From the above plots, early observations suggests that all listed actions are almost distinct with the exception of *"my"* and *"your"*. The similarity of these two actions can be seen in Figure 14.

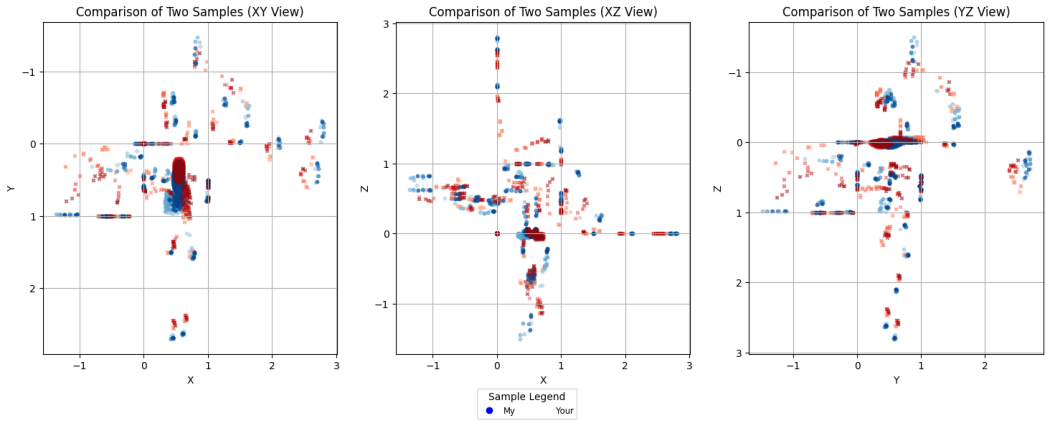
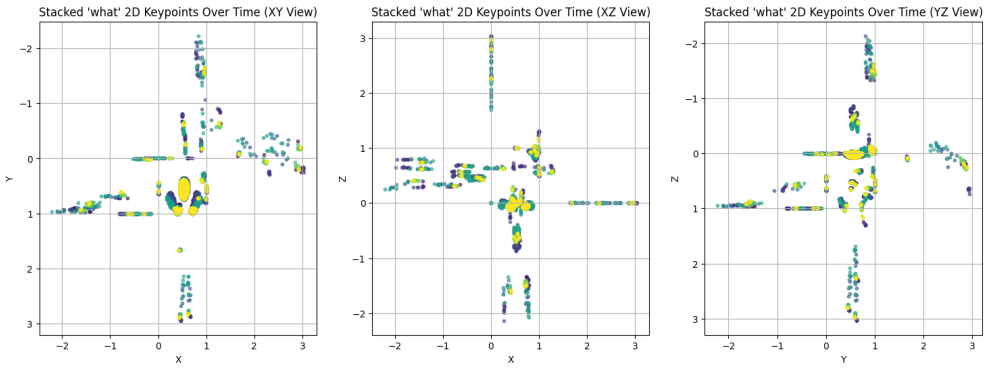


Figure 14: "my" and "your" action sequences are similar

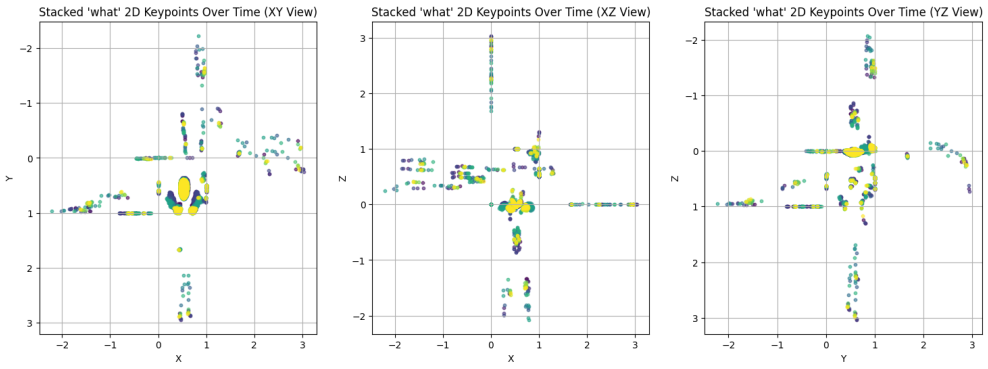
5.2 Data Pre-processing

5.2.1 Frame Skipping Mechanism

To reduce computational costs and improve the model's generalisability, a frame-skipping mechanism was implemented. This method involves sampling frames at regular intervals, to mitigate overfitting to the training set (Muhammad et al., 2023). By doing so, the model not only reduces noise but also learns to recognise key temporal features rather than memorising precise frame sequences, leading to improved performance on unseen data (refer to Figure 15).



(a) Without frame skipping for *what* sequence



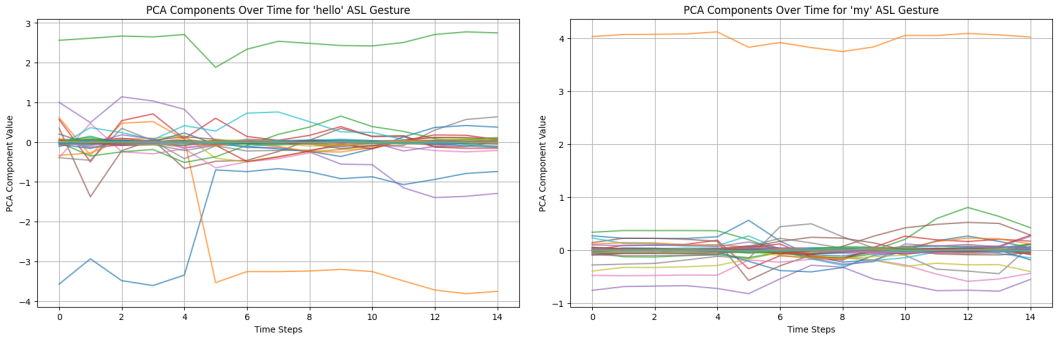
(b) With frame skipping of 1 for *what* sequence

Figure 15: Frame skipping reduces input noise and dimensionality, while maintaining the essential temporal structure of the observed action.

5.2.2 Principal Component Analysis

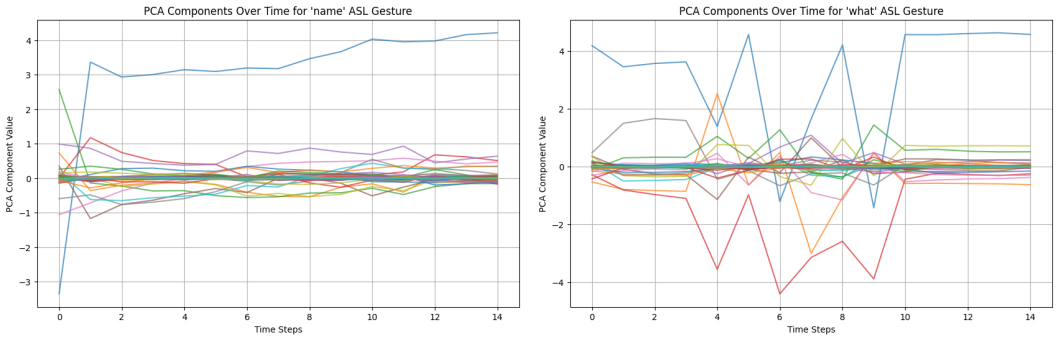
It was noted that the dimension of the feature set was 1662. This is significantly larger than the number of samples collected for this project. This would hence in practice result in model training to overfit the noise in the sample set. PCA was hence performed for dimensional reduction.

5.2.2.1 Initial Observations



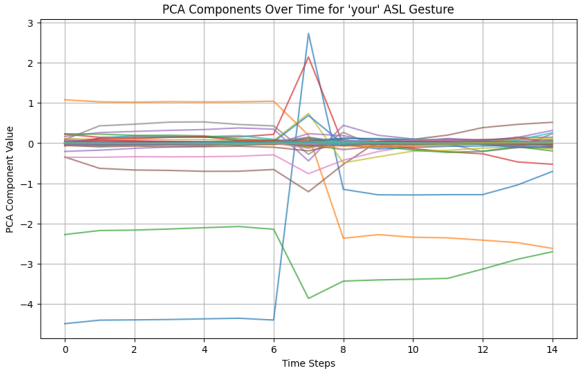
(a) “hello”

(b) “my”



(c) “name”

(d) “what”



(e) “your”

Figure 16: PCA-Reduced Visualisations of Sign Language Actions for 50 principal components

50 PCA components were initially used and a PCA component test was performed and plotted (refer to Figure 16). It can be observed that many components hover around zero, indicating they contribute very little variance to the representations. Hence, many of such components are inessential and likely represents noise or redundant information.

5.2.2.2 Picking Number of Principal Components

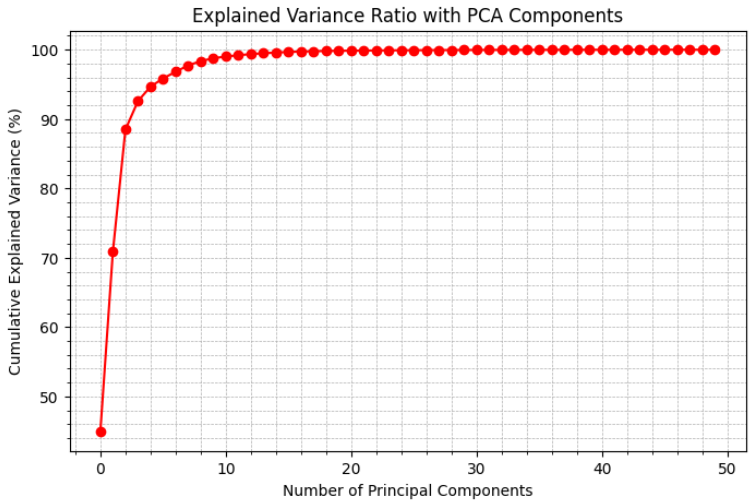


Figure 17: Explained Variance Ratio plot against the Number of Principal Components

From the explained variance ratio plot (refer to Figure 17), the first few components capture most of the variance. At around 5 components, the plot shows that it explains nearly 95% of the variance. After about 10 components, the additional variance captured flattens out, meaning extra components contribute less additional information. Hence, a PCA of 10 components was chosen giving 98.76% retained variance.

5.2.2.3 Observations with 10 PCA Components

A plot of the PCA component values with time for 10 components can be found in Figure 18. It was noted that many components, beyond 5 components, appear relatively flat across all actions. This suggests that most of the meaningful variance is captured within the first few principal components.

Additionally, the similarity between *"my"* and *"your"* makes it hard to differentiate which may potentially be an issue during model training (PCA 6 to PCA 10). In contrast, *"hello"* and *"what"* showed greater fluctuations over time providing greater differentiation.

Overall, the PCA trends displays temporal consistency. Despite some variations, most PCA components remain stable over time, reinforcing the notion that key movement patterns persist across time steps. This stability reinforces the suitability for these transformed features for training.

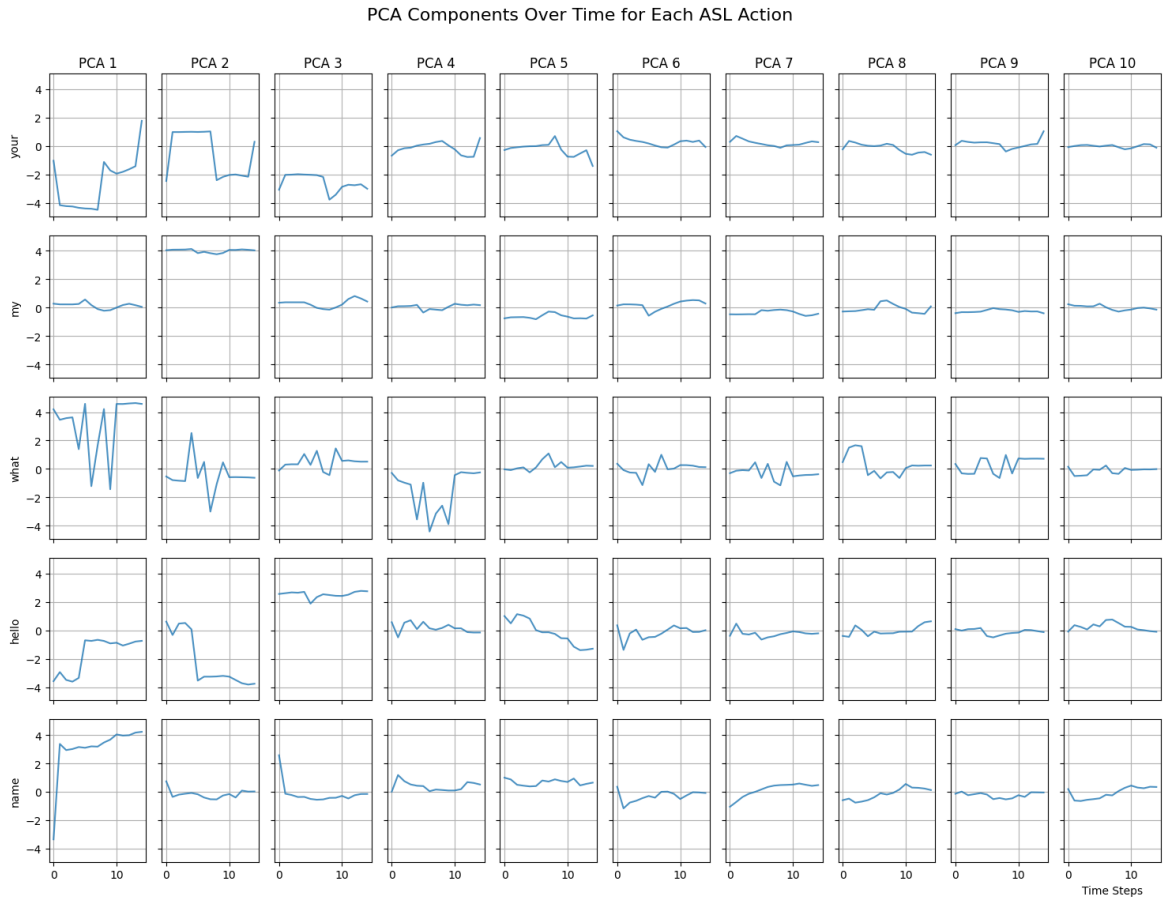


Figure 18: PCA-Reduced Visualisations of Sign Language Actions for 10 PCA components

5.2.2.4 PCA Conclusion

The reduction of dimensions to 10 PCA dimensions is beneficial to the small dataset collected for this project (at most 180 sequences per action). The original feature space of 1662 dimensions per frame can lead to overfitting. The reduction significantly lowers model complexity, and hence reduce overfitting risk.

5.2.3 Sample Generation using Augmentation

Additional training samples were generated using data augmentation techniques. The applied transformations included:

- Applying random scaling → Sampled from Uniform Distribution (0.9 to 1.1)
- Applying random translations → Sampled from Uniform Distribution (-0.05 to 0.05)
- Applying random rotations → Sampled from Uniform Distribution (-5, 5 degrees)

These transformations introduce controlled variability in the dataset which provides more differentiation in motion which can help improve model generalisability across unseen sign variations (refer to Figure 20).

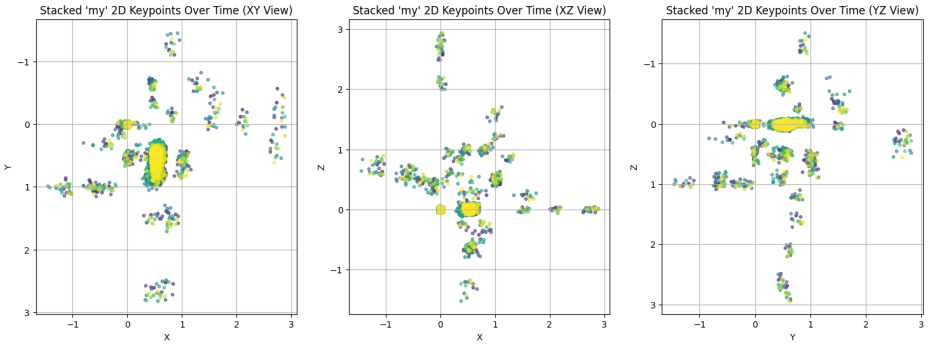


Figure 19: Augmented Data for action "my""

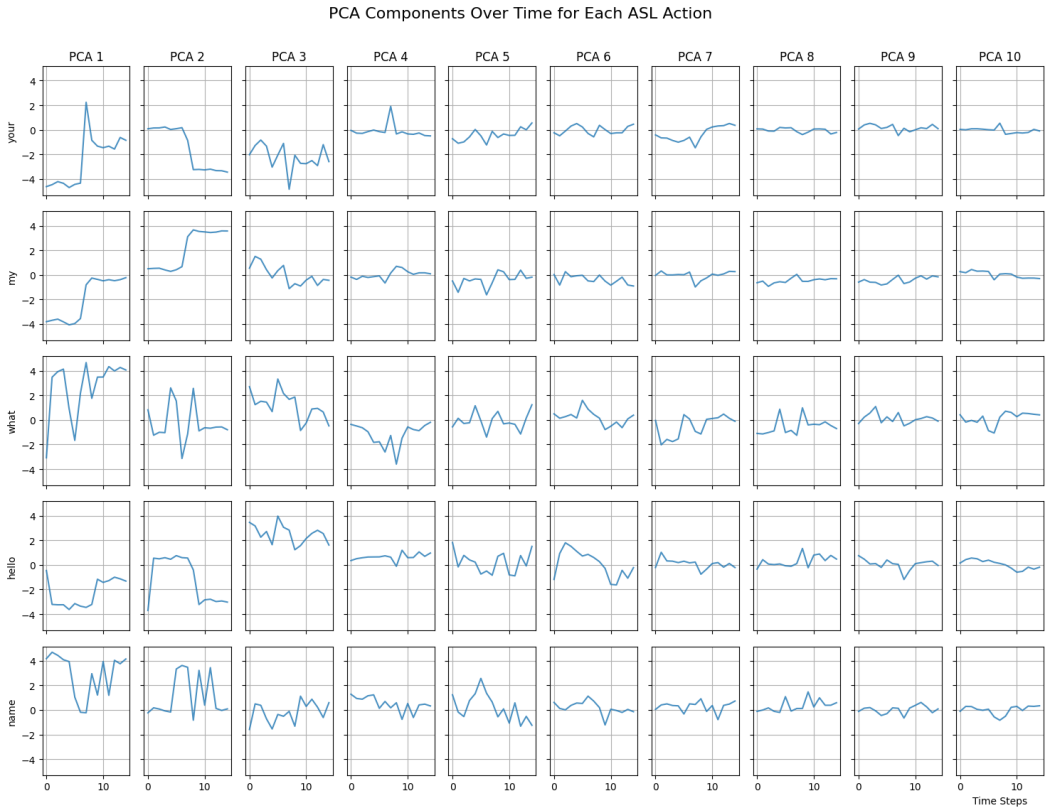


Figure 20: PCA-Reduced Visualisations for Augmented Dataset for 10 PCA components

5.2.4 Data Train/Validation/Test Split

The dataset was split into training, validation, and test sets using the following proportions:

- 80% for training, used to optimise model parameters.
- 20% for testing, reserved for evaluating final model performance on unseen data.
- 10% of the training set for validation, used for hyperparameter tuning and preventing overfitting.

This split ensures that the model learns effectively while retaining a sufficient amount of data for unbiased evaluation.

5.2.4.1 Stratified Sampling

To maintain a balanced dataset, stratified sampling was used during train-test split. In stratified sampling, data is split in a way that preserves the original class distribution across the training, validation, and test sets. This prevents under representation of classes due to random sampling.

By using stratified sampling, each subset (train, validation, and test) would contain the same proportion of each class as in the full dataset. This prevents the model from being biased toward the more frequent classes and improves generalisation to new data.

5.3 Model

5.3.1 Model Overview

The input to the model consists of collated landmarks, which are the flattened NumPy representations of the keypoints extracted by MediaPipe. These arrays preserve the temporal dimension of the data and serve as the sequential input features to the network.

The backbone of the model comprises stacked LSTM layers, which are well-suited for modelling temporal dependencies. Multiple LSTM layers allow the model to extract increasingly abstract sequence representations, making them effective for gesture and action recognition tasks.

Building upon prior architectural insights, an inverse hourglass-shaped architecture is proposed for experimentation. This structure either expands the feature dimensionality before contracting it or applies a purely contracting design. The rationale for such a configuration stems from its potential to enhance the model's capacity to capture fine-grained temporal dynamics during the expansion phase, followed by compression to enforce generalisation and reduce overfitting (Benhaili et al., 2022). However, this remains a proposed design and will be validated empirically during experimentation.

Additionally, dense layers are proposed after the LSTM blocks, as inspired by recent studies in action recognition (Sateesh et al., 2024). These layers introduce non-linear transformations, enabling richer feature interactions before the final classification step. The model concludes with a softmax layer, which outputs class probabilities corresponding to the ASL gestures.

5.3.2 Model Training and Optimisation

To improve the model's ability to generalise across different action sequences, the dataset was augmented (refer to Section 5.2.3) to increase the overall number of training samples. This enhancement improves the overall robustness of the model within each action class.

Training was conducted using a learning rate of `0.00005`, chosen based on empirical observations to reduce overfitting. A smaller learning rate allowed for more stable convergence and finer weight updates during training.

The action index used for classification is shown in Table 8, mapping each gesture to its corresponding label index and sample count after augmentation.

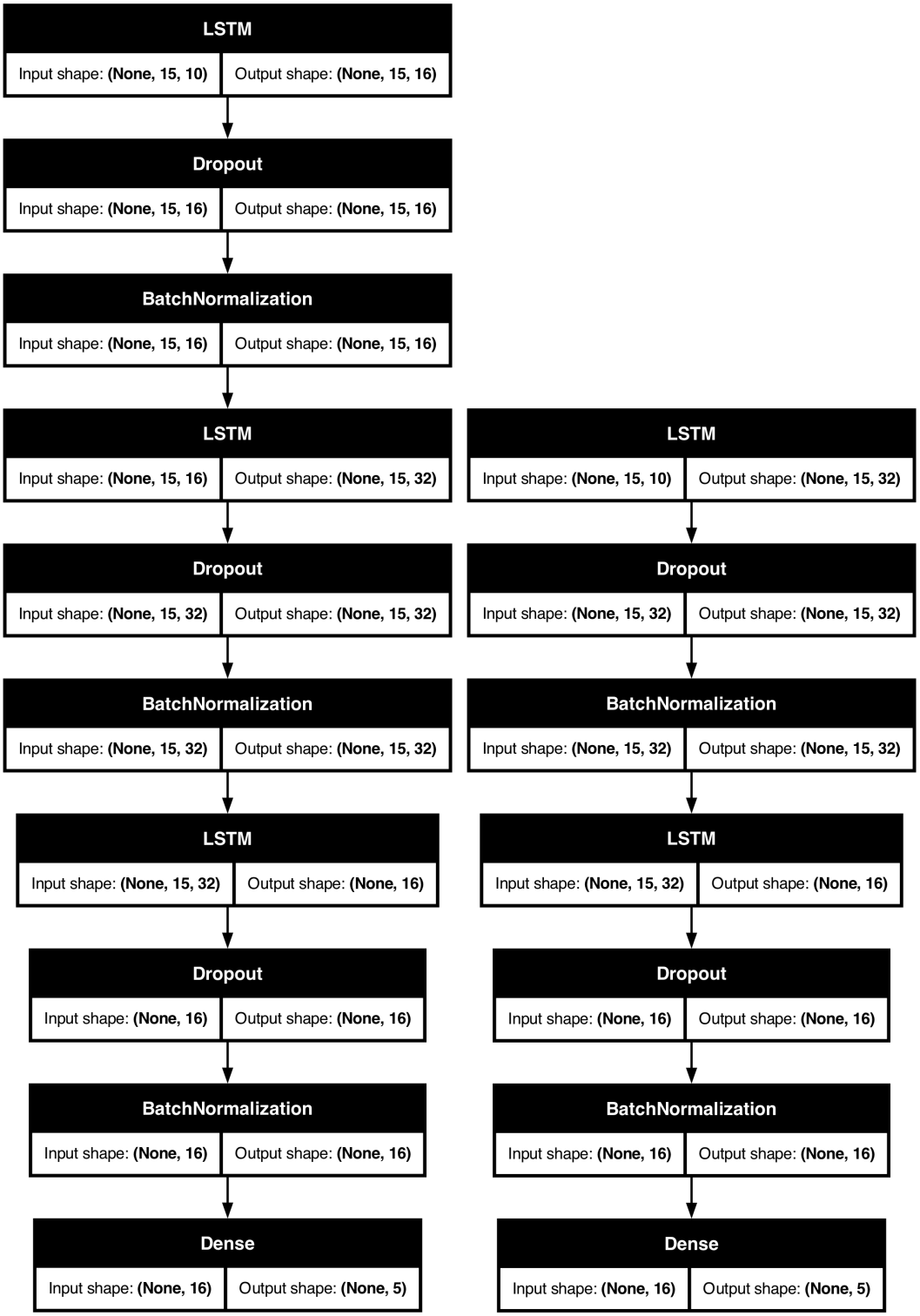
Action	Index	Sample Count
your	0	900
my	1	900
what	2	1080
hello	3	1080
name	4	900

Table 8: Action index table

5.3.3 LSTM 16/32/16 and LSTM 32/16 Layers

5.3.3.1 Model Architecture

The dataset was trained using Adam optimiser on two model configurations: LSTM 16/32/16 and LSTM 32/16, as shown in Figure 21.



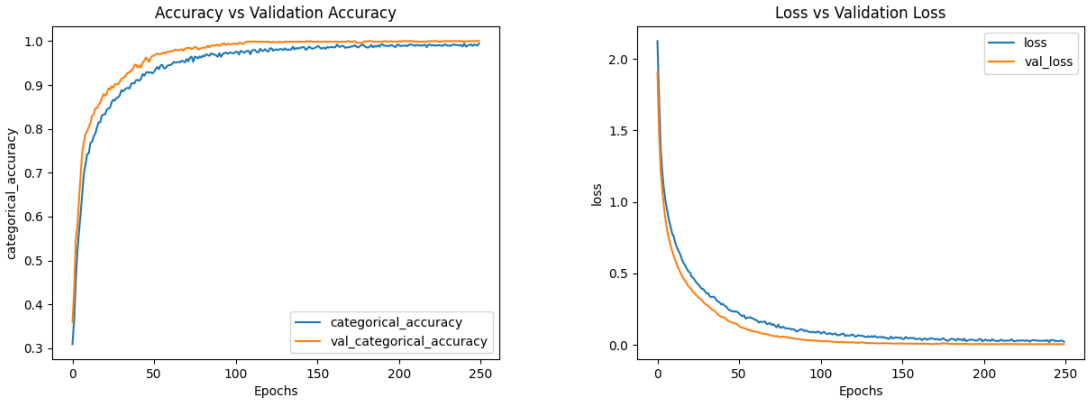
(a) LSTM 16/32/16 (b) LSTM 32/16

Figure 21: Model Summary for LSTM 16/32/16 and LSTM 32/16

5.3.3.2 Results

LSTM 16/32/16

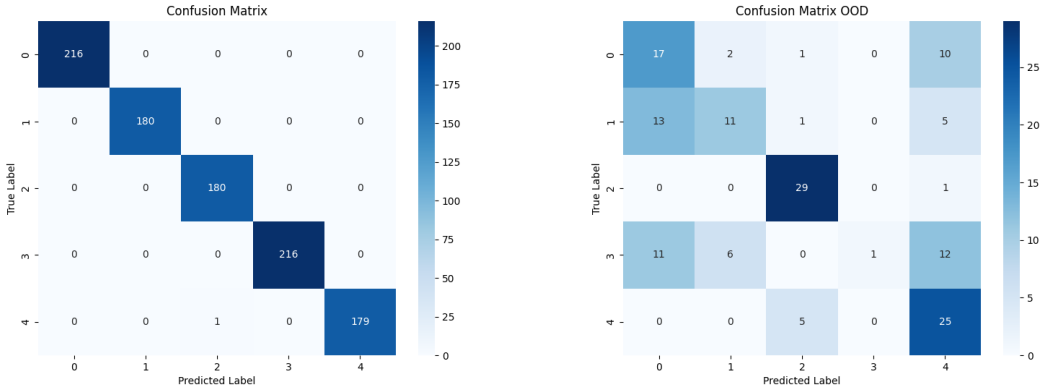
Figure 22 presents the accuracy and loss curves for the LSTM 16/32/16 model over 250 epochs.



(a) Accuracy (b) Categorical Loss

Figure 22: Accuracy and Loss Curves for LSTM 16/32/16 for 250 epochs

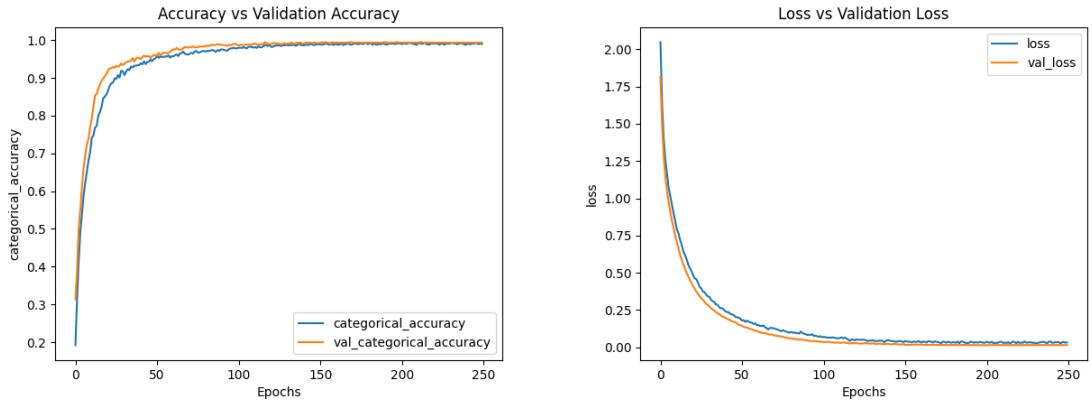
Figure 23 shows the confusion matrix at 150 epochs.



(a) Confusion Matrix (b) Confusion Matrix (Out-of-domain)

Figure 23: Confusion Matrix for LSTM 16/32/16 for 150 epochs

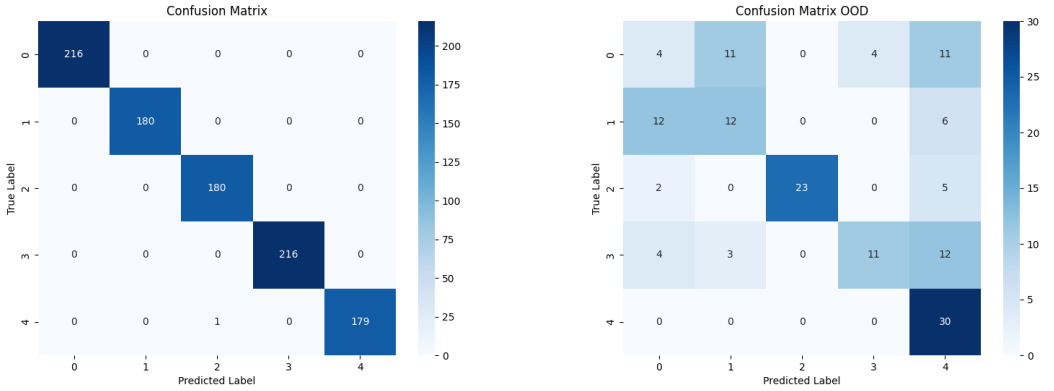
LSTM 32/16
Figure 24 presents the accuracy and loss curves for the LSTM 32/16 model.



(a) Accuracy (b) Categorical Loss

Figure 24: Accuracy and Loss Curves for LSTM 32/16 for 250 epochs

Figure 25 shows the confusion matrix at 150 epochs.



(a) Confusion Matrix (b) Confusion Matrix (Out-of-domain)

Figure 25: Confusion Matrix for LSTM 32/16 for 150 epochs

5.3.3.3 Evaluation

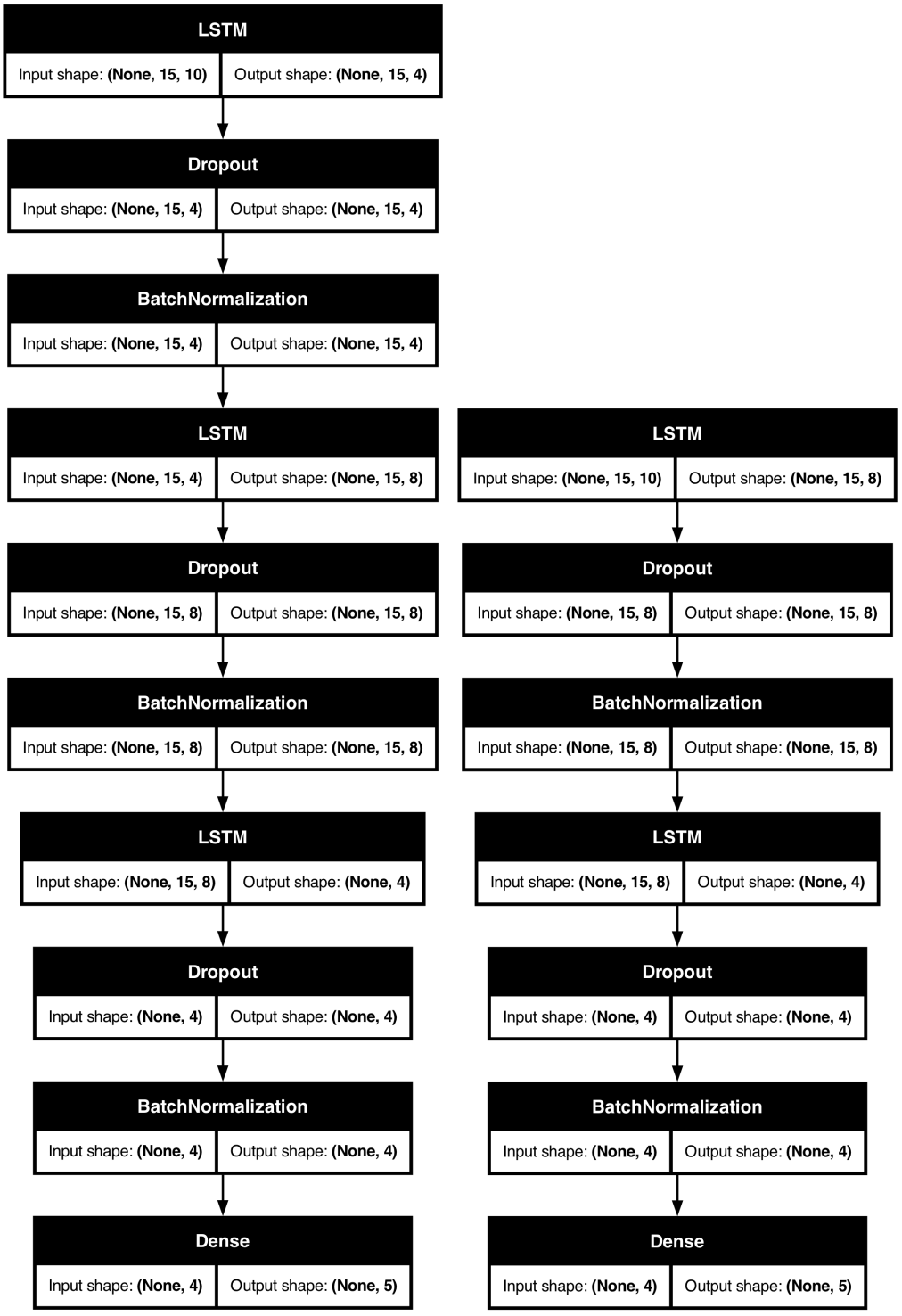
Both models exhibit a steep improvement in accuracy and loss within the first 50 epochs, suggesting the model may be overparameterised for the given training data. The confusion matrix further indicates overfitting to the in-domain dataset, with near-perfect classification on the test set at only 150 epochs.

However, performance deteriorates significantly on the out-of-domain dataset, demonstrating poor generalisation. The model frequently confuses *"your"*`(0)` and *"my"*`(1)`, as well as *"hello"*`(3)` with *"name"*`(4)`, indicating weaknesses in feature differentiation. These results suggest that regularisation or reducing model complexity is necessary to improve robustness.

5.3.4 LSTM 4/8/4 and LSTM 8/4 Layers

5.3.4.1 Model Architecture

To address severe overfitting, which resulted in a highly biased model towards the in-domain dataset, the model complexity was significantly reduced. The revised architecture is shown in Figure 26.

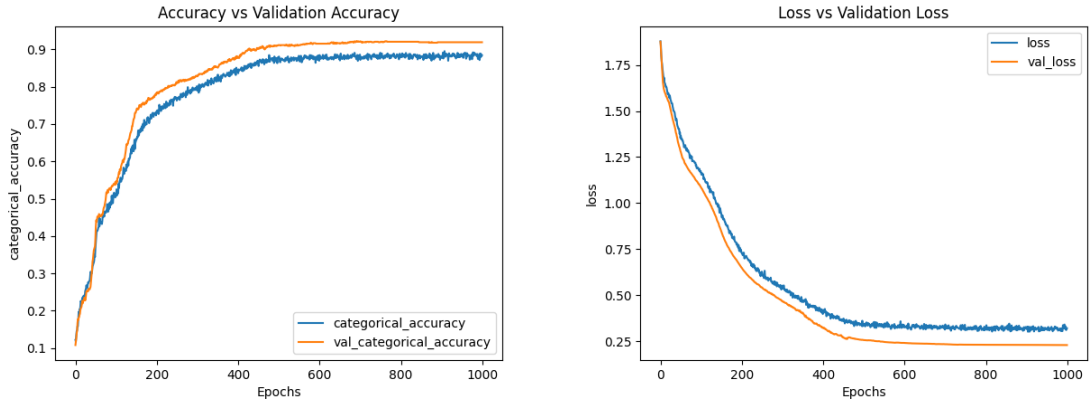


(a) LSTM 4/8/4 (b) LSTM 8/4

Figure 26: Model Summary for LSTM 4/8/4 and LSTM 8/4

5.3.4.2 Results

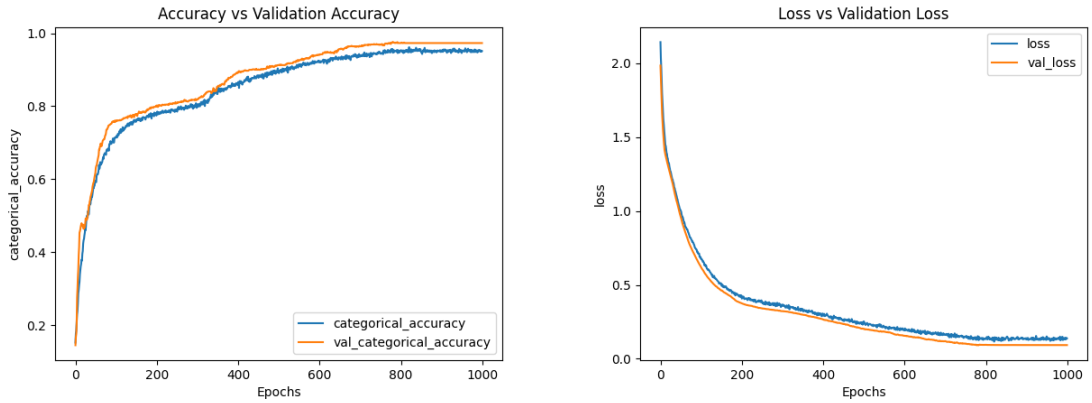
LSTM 4/8/4
Figure 27 presents the accuracy and loss curves for the model over 1000 epochs.



(a) Accuracy (b) Categorical Loss

Figure 27: Accuracy and Loss Curves for LSTM 4/8/4

LSTM 8/4
Figure 28 presents the accuracy and loss curves for the model over 1000 epochs.



(a) Accuracy (b) Categorical Loss

Figure 28: Accuracy and Loss Curves for LSTM 8/4

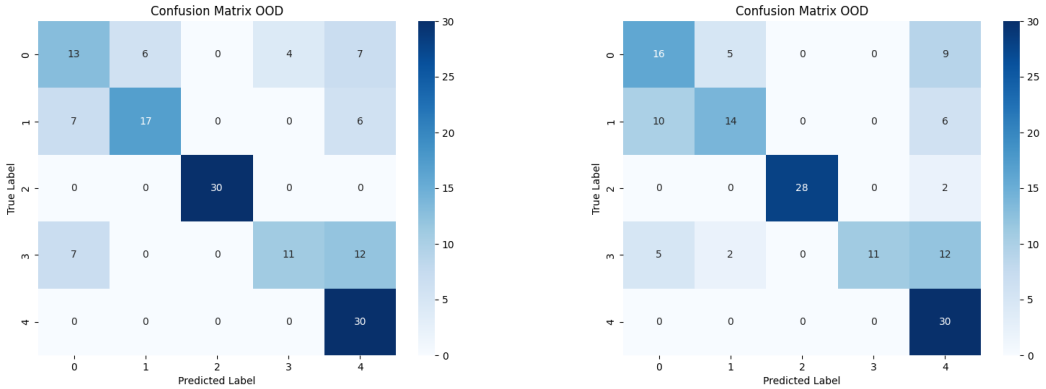
5.3.4.3 Evaluation

Table 9 presents the best Macro-F1 scores achieved by each model configuration, along with the corresponding epoch at which the score was obtained.

Model Configuration	**Best Macro-F1 Score**	Epoch
LSTM 4/8/4	0.658484	1000
LSTM 8/4	0.649974	800
LSTM 32/16	0.646218	1000
LSTM 16/32/16	0.636999	650

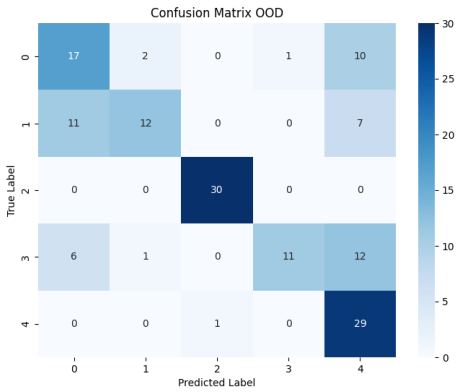
Table 9: Best Macro-F1 scores based on three runs for each model configuration.

Figure 29 presents the confusion matrices for out-of-domain performance across the different best model configurations.

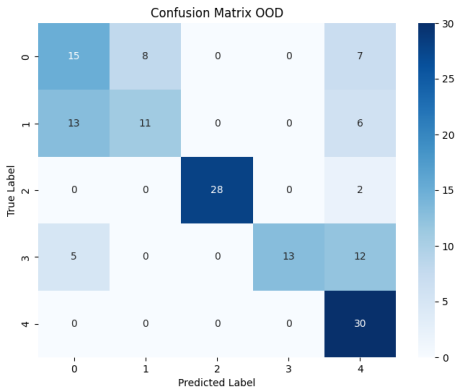


(a) LSTM 4/8/4

(b) LSTM 8/4



(c) LSTM 32/16



(d) LSTM 16/32/16

Figure 29: Confusion matrices for out-of-domain evaluation across different best model configurations.

The confusion matrices indicate that all models, regardless of complexity, exhibit similar classification performance on the out-of-domain dataset. This suggests that reducing model complexity does not negatively impact generalisation. The LSTM 4/8/4 and LSTM 8/4 models achieve comparable results to the more complex LSTM 16/32/16 and LSTM 32/16 models, demonstrating that a simpler architecture is sufficient for this classification task.

Additionally, the accuracy and loss curves show that simpler models continue improving until approximately 500 epochs, whereas more complex models (LSTM 16/32/16 and LSTM 32/16) stagnate much earlier, around 50 epochs. This suggests that the simpler models have a more appropriate level of parameterisation, allowing them to gradually learn patterns rather than memorising the dataset too quickly. The early stagnation of larger models implies that they may be overfitting, leading to minimal gains despite higher complexity.

The performance consistency across models suggests that simpler architectures effectively mitigate overfitting while retaining the ability to differentiate between key classes. The classification of *"your"*`(0)`, *"my"*`(1)`, and *"what"*`(2)` remains stable across different architectures, indicating that feature representations are robust. Furthermore, since increasing model complexity does not provide a significant performance boost, this reinforces that a more compact model can generalise just as effectively while reducing computational costs.

Overall, these findings indicate that simpler architectures (LSTM 4/8/4 and LSTM 8/4) are preferable, as they achieve better parameter efficiency and long-term generalisation while being computationally efficient. This supports the argument that optimising model size is crucial for balancing performance and efficiency in real-world applications.

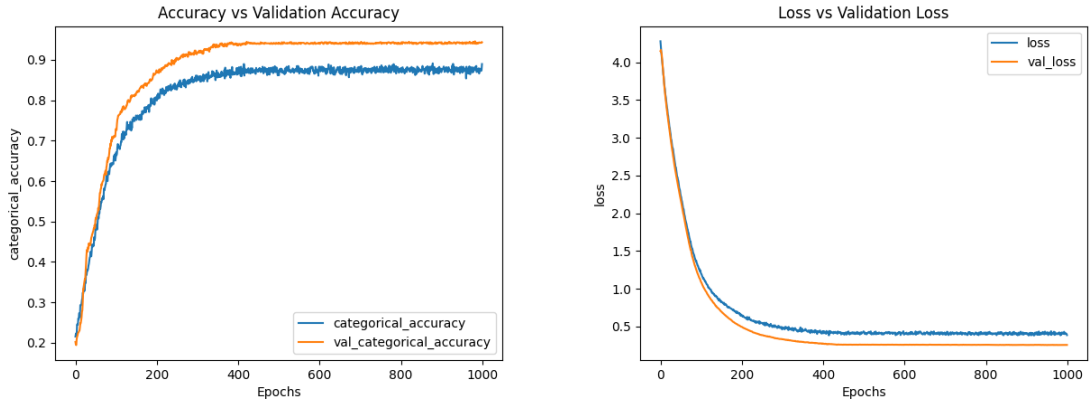
5.3.5 LSTM 4/8/4 and LSTM 8/4 Layers with L2-Regulariser

5.3.5.1 Model Architecture

To address remaining signs of overfitting and further refine the model, L2 regularisation with a factor of $\lambda = 0.05$ was applied to each LSTM layer, aiming to improve generalisation while maintaining model simplicity.

5.3.5.2 Results

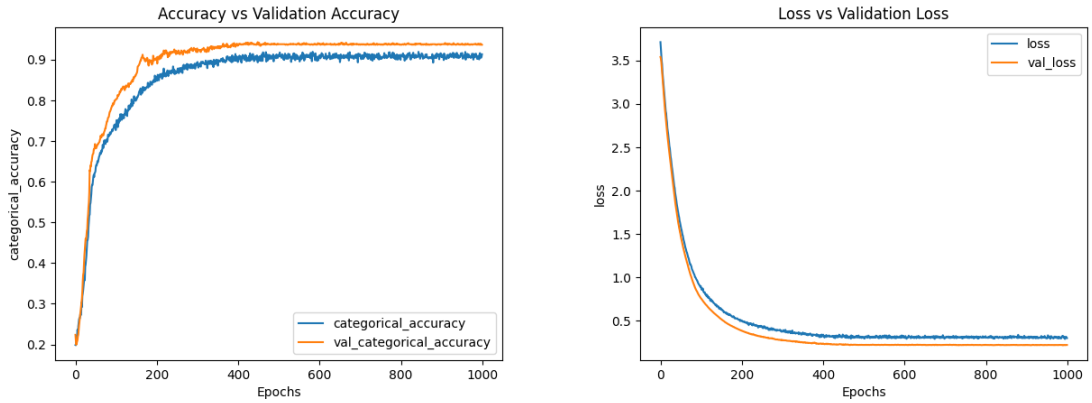
LSTM 4/8/4 with L2-Regulariser ($\lambda = 0.05$)
Figure 30 presents the accuracy and loss curves for the model over 1000 epochs.



(a) Accuracy (b) Categorical Loss

Figure 30: Accuracy and Loss Curves for LSTM 4/8/4 with L2-Regulariser ($\lambda = 0.05$) for 1000 epochs

LSTM 8/4 with L2-Regulariser ($\lambda = 0.05$)
Figure 31 presents the accuracy and loss curves for the model over 1000 epochs.



(a) Accuracy (b) Categorical Loss

Figure 31: Accuracy and Loss Curves for LSTM 8/4 with L2-Regulariser ($\lambda = 0.05$) for 1000 epochs

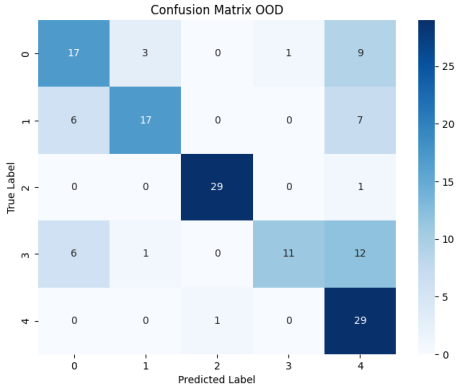
5.3.5.3 Evaluation

Table 10 presents the best Macro-F1 scores achieved by each model configuration, along with the corresponding epoch at which the score was obtained.

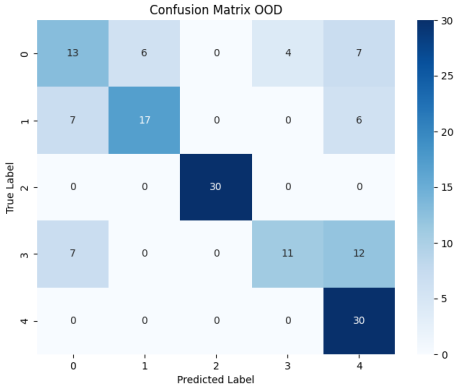
Model Configuration	**Best Macro-F1 Score**	Epoch
LSTM 4/8/4 with L2-Regularisation ($\lambda = 0.05$)	0.678501	750
LSTM 4/8/4	0.658484	1000
LSTM 8/4	0.649974	800
LSTM 8/4 with L2-Regularisation ($\lambda = 0.05$)	0.648004	850

Table 10: The best Macro-F1 Score based on three runs for each model configuration

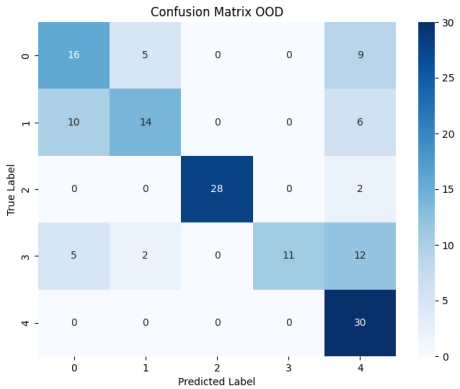
Figure 32 presents the confusion matrices for out-of-domain performance across different model configurations.



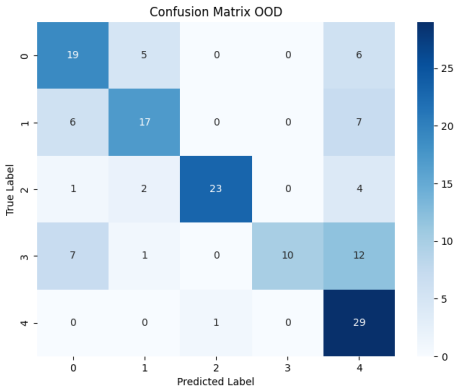
(a) LSTM 4/8/4 L2-Regularisation ($\lambda = 0.05$)



(b) LSTM 4/8/4



(c) LSTM 8/4



(d) LSTM 8/4 L2-Regularisation ($\lambda = 0.05$)

Figure 32: Confusion matrices for out-of-domain evaluation across different model configurations.

The results show that applying L2 regularisation ($\lambda = 0.05$) has a modest but positive effect on training efficiency and performance. The LSTM 4/8/4 model with regularisation achieved the highest Macro-F1 score (`0.6785`) and converged faster (at `750` epochs) than its non-regularised counterpart (`0.6585` at `1000` epochs). In contrast, for the smaller LSTM 8/4 model, regularisation had little to no impact, suggesting that it is more beneficial in larger models where it helps stabilise training.

When comparing architectures, the LSTM 4/8/4 model consistently outperformed the 8/4 variant in both performance and stability. This may be attributed to its reverse-hourglass structure, which incorporates an expansion followed by compression phase (Benhaili et al., 2022). Such a structure

likely promotes the extraction of compact but expressive temporal features, aiding generalisation. The 8/4 model, lacking this expansion, may be more limited in capturing high-level patterns.

Additionally, the 4/8/4 model demonstrated a more gradual and sustained learning curve, continuing to improve up to `1000` epochs, whereas the 8/4 model plateaued around `800` epochs. This prolonged learning suggests that hierarchical feature abstraction in the reverse-hourglass design contributes to better long-term optimisation. While these observations align with prior research on hourglass architectures in action recognition, further experiments are needed to isolate the impact of architecture shape from other contributing factors.

5.3.6 LSTM 8/4 with Dense 4

5.3.6.1 Model Architecture

Dense layer was added in hopes that non-linear transformations may improve feature interactions (Sateesh et al., 2024). Figure 33 shows the updated model architecture.

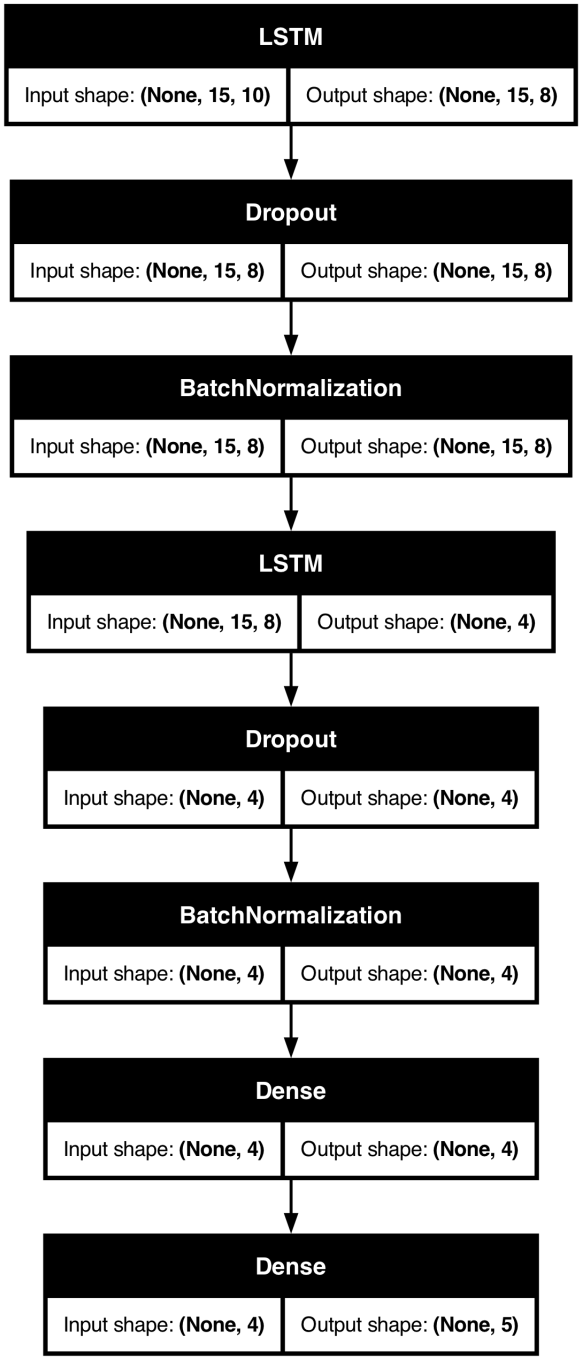
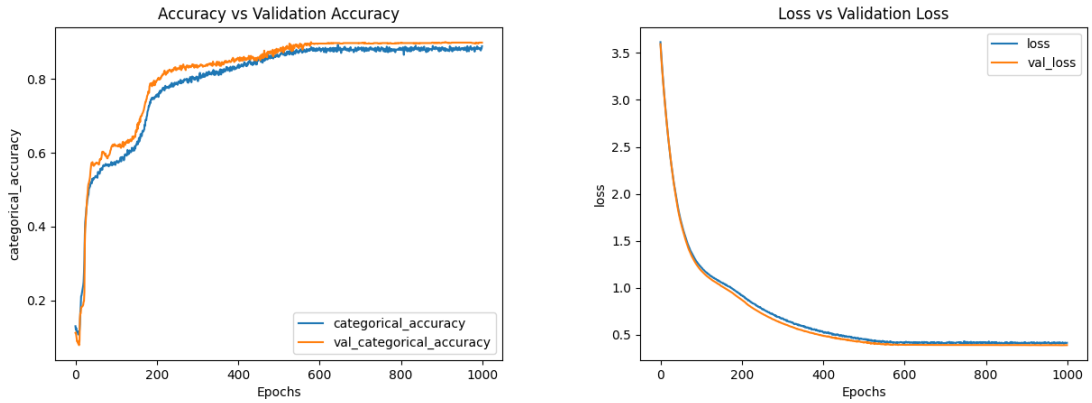


Figure 33: Model Summary for LSTM 8/4 with Dense 4

5.3.6.2 Results

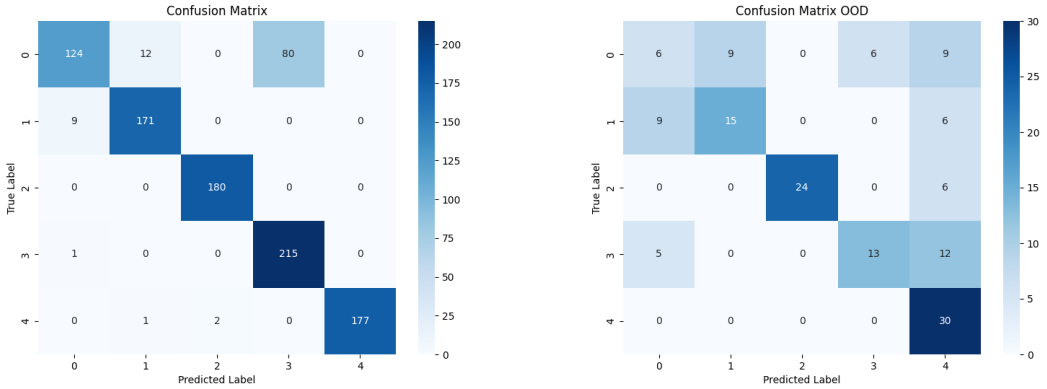
Figure 34 presents the accuracy and loss curves for the model over 1000 epochs.



(a) Accuracy (b) Categorical Loss

Figure 34: Accuracy and Loss Curves for LSTM 8/4 with Dense layer for 1000 epochs

Figure 35 shows the confusion matrix at 1000 epochs.



(a) Confusion Matrix (b) Confusion Matrix (Out-of-domain)

Figure 35: Confusion Matrix for LSTM 8/4 with Dense Layer for 1000 epochs

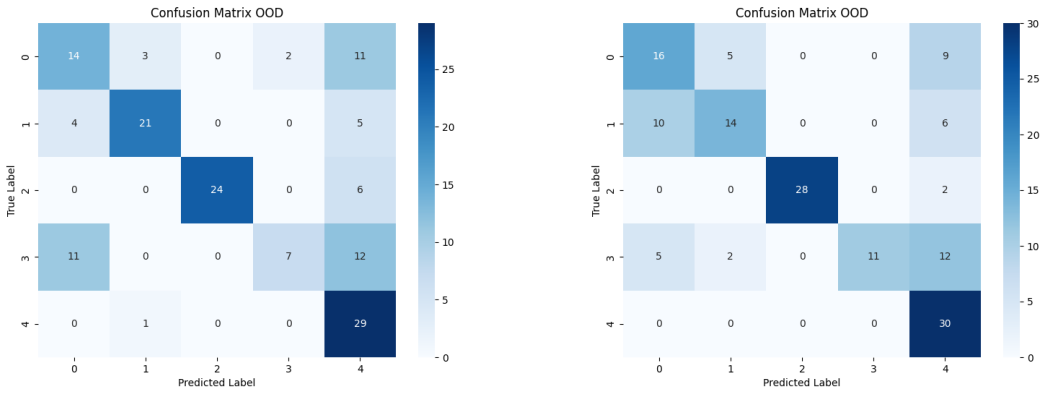
5.3.6.3 Evaluation

Table 11 presents the best Macro-F1 scores achieved by each model configuration, along with the corresponding epoch at which the score was obtained.

Model Configuration	**Best Macro-F1 Score**	Epoch
LSTM 8/4	0.649974	800
LSTM 8/4 with Dense 4	0.621946	550

Table 11: Best Macro-F1 scores based on three runs for LSTM 8/4 and LSTM 8/4 with Dense 4

Figure 36 presents the confusion matrices for out-of-domain performance across both configurations.



(a) LSTM 8/4 with Dense 4 output layer

(b) LSTM 8/4

Figure 36: Confusion matrices for out-of-domain evaluation of LSTM 8/4 with and without an additional Dense 4 layer

The results indicate that adding a dense layer at the output does not enhance model performance and may, in fact, slightly degrade generalisation. The LSTM 8/4 model without the dense layer achieved a higher Macro-F1 score (`0.6500`) at `800` epochs, while the LSTM 8/4 with Dense `4` peaked at `0.6219` at `550` epochs. This suggests that while the additional dense layer may accelerate initial training, it does not contribute to long-term performance gains.

The confusion matrices further reinforce this observation, as both models exhibit similar misclassification patterns, with no significant reduction in errors for the model with the dense layer. The LSTM 8/4 model without the dense layer demonstrated more stable training across epochs, implying that the extra dense layer introduces unnecessary complexity without improving feature representation for this classification task.

These findings suggest that in the context of LSTM-based action recognition, additional dense layers at the final stage may not necessarily improve performance. Instead, they may lead to increased parameter redundancy, requiring more training data to be effective. The simpler LSTM 8/4 model remains the preferred configuration, balancing efficiency and generalisation without unnecessary architectural overhead.

5.3.7 Candidate Model

Based on the evaluation results, the LSTM 4/8/4 model with L2-Regularisation ($\lambda = 0.05$) trained for `750` epochs was selected as the best-performing configuration. Its confusion matrices, presented in Figure 32a, illustrate its classification performance on the out-of-domain datasets.

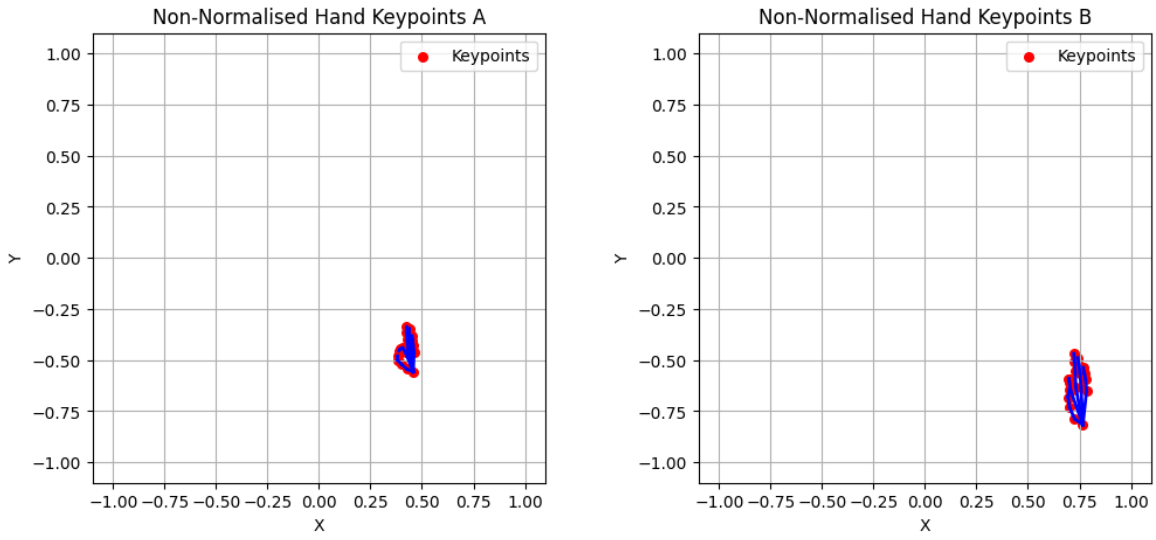
6 Alphabet Recognition

This section explores the use of FNN in classifying ASL alphabets and numbers for the static gesture classification component of the proposed model.

6.1 Data Exploration

The hand keypoints were extracted to provide a structured representation of finger positions—essential for alphabet recognition. Additionally, only the (x, y) coordinates were recorded, as the z-coordinate was not crucial for distinguishing static gestures. This 2D spatial information was sufficient for classification, resulting in a 42-dimensional feature set.

Each collected dataset was then visualised to better understand the distribution of keypoints. It was observed that the absolute positions of keypoints varied across samples, as seen in Figure 37, which could introduce unnecessary noise into the model.



(a) Sample A of label 'F'

(b) Sample B of label 'F'

Figure 37: Example of varied hand placements in collected data

For static gesture recognition, the absolute position of the hand within the frame is not essential. Instead, the model benefits more from the relative positioning of the fingers, highlighting the need for the data to be pre-processed before being fed to the model.

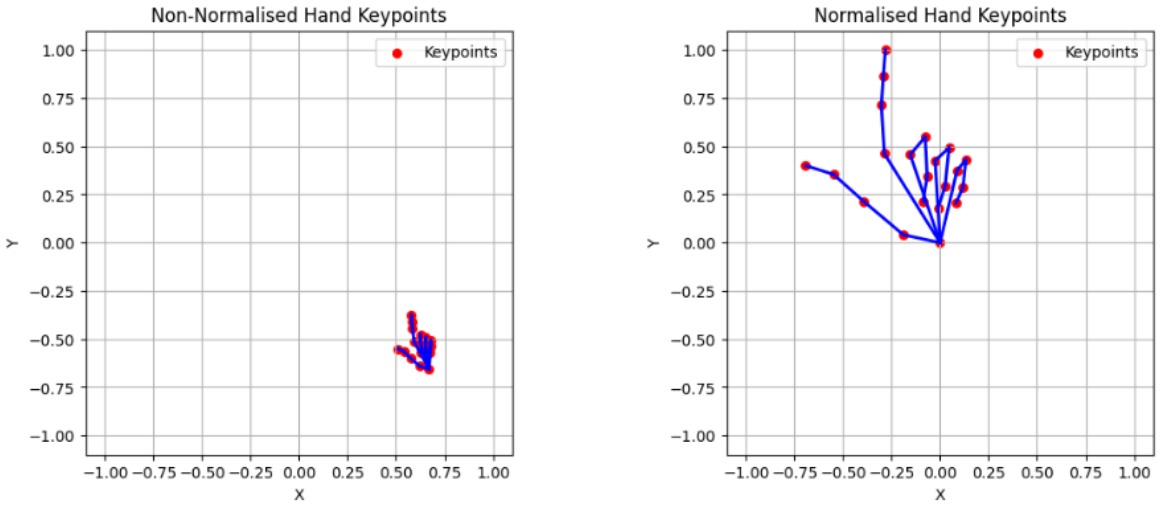
6.2 Data Pre-processing

6.2.1 Keypoint Normalisation

The extracted hand landmarks were pre-processed to ensure consistency and robustness of the model. Transformations included:

- **Conversion to relative coordinates:**
  The first landmark (wrist) is used as a reference point, and all other keypoints are adjusted relative to it. This eliminates variations caused by hand placement within the frame, ensuring that the hand's absolute position in the image does not affect feature values.
- **Normalisation:**
  Each coordinate is divided by the largest absolute coordinate value in the frame. This scales all values between -1 and 1, ensuring uniformity in data representation, maintaining consistency across different hand sizes and camera distances.

As a result, the hand is always mapped to a fixed coordinate space, but its shape and proportions remain intact.



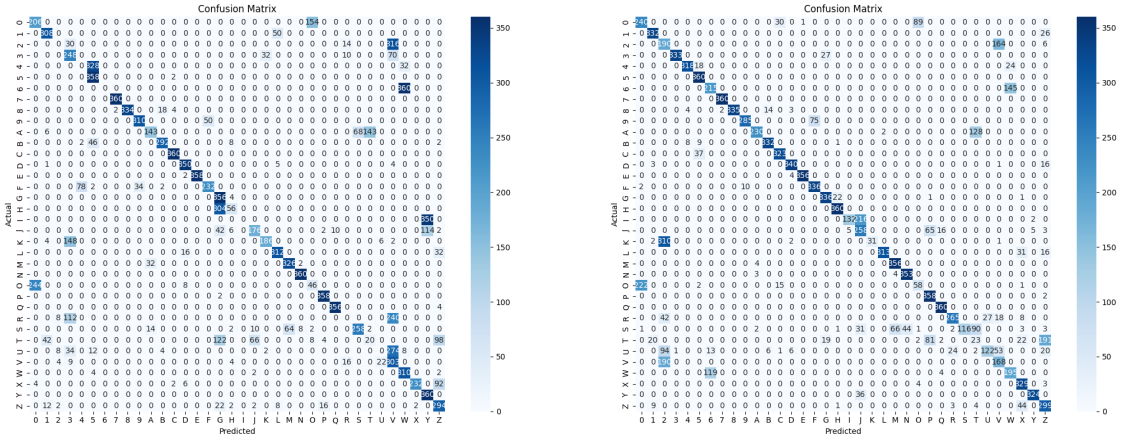
(a) Raw keypoints (Letter 'L')

(b) Normalised keypoints (Letter 'L')

Figure 38: Raw vs normalised keypoints

6.2.1.1 Results of Data Normalisation

The accuracy of the model was evaluated using its corresponding out-of-domain datasets. Two assessments were conducted: one using raw data and the other using normalised data to compare performance variations. A consistent FNN model with Dense 64/32 architecture was used across both evaluations to ensure a fair comparison.



(a) Confusion matrix of raw data classification

(b) Confusion matrix of normalised data classification

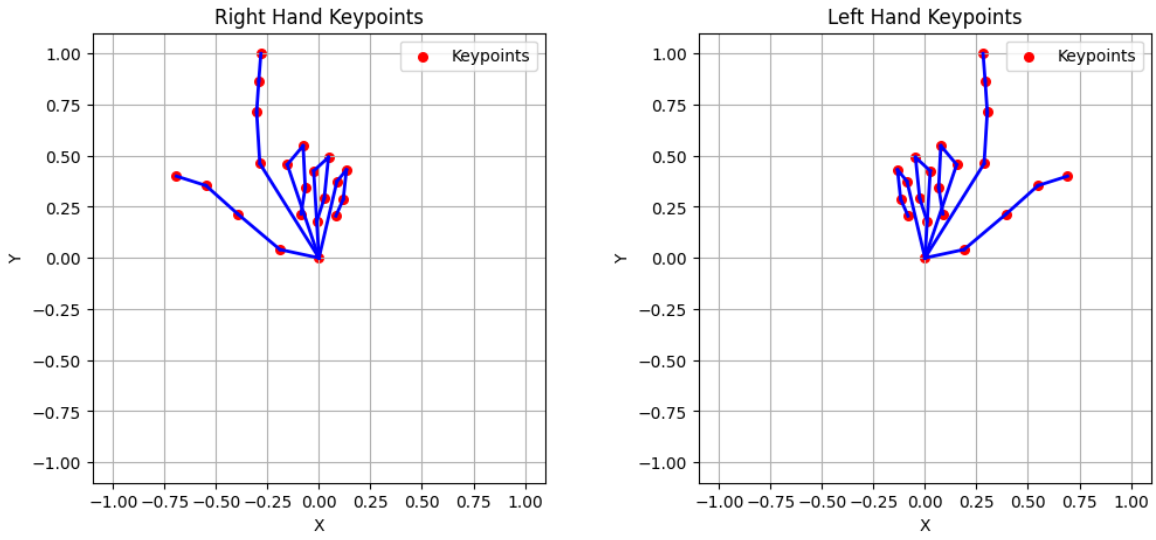
Figure 39: Raw vs normalised model performance on out-of-domain data

Despite accurate classification for most of the labels, it was observed that the model trained on raw data tends to misclassify some gestures more frequently than the model trained on normalised data.

As a result, normalisation can enhance the model's ability to generalise different participants and scenarios, leading to more reliable gesture recognition.

6.2.2 Sample Generation

The initial data collection captured only the keypoints of the right hand. To enable the model to recognise gestures from both hands, the right-hand keypoints were reflected to simulate left-hand counterparts. This approach effectively doubled the dataset size and ensured the model could perform equally well on both hands.



(a) Right hand keypoints of letter 'L'

(b) Left hand keypoints of letter 'L'

Figure 40: Reflected keypoints

As such, the resulting sample count after generation can be seen in Table 12:

Action	Number of datasets per label
A-Z, 0-9	2,400

Table 12: Sample count after generation

6.2.3 Data Train/Test Split

To ensure a robust evaluation of our model, the dataset was split into training, test and validation sets using the following proportions:

- 75% for training.
- 25% for testing.
- 10% of training for validation.

This split ensures that the model learns effectively while retaining a sufficient amount of data for unbiased evaluation.

6.3 Model

6.3.1 Model Overview

For hand gesture recognition based on per-frame keypoint data, a FNN was employed. FNNs are commonly chosen for their simplicity and efficiency in processing structured input data. Specifically, they excel in handling hand-crafted features derived from keypoint coordinates, enabling effective classification of gesture patterns (Reddy and Anitha, 2022).

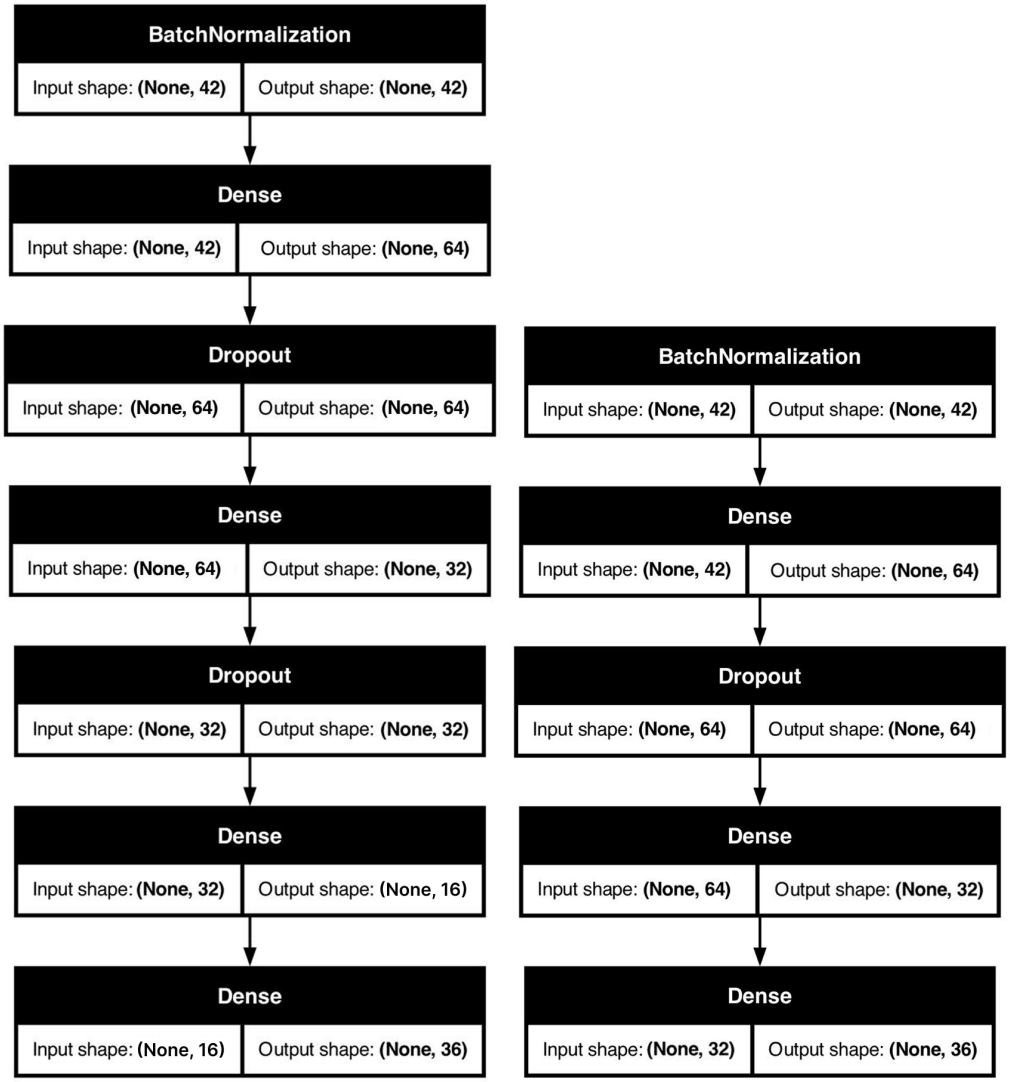
To improve learning capacity and gradient flow, the Mish activation function was integrated into

the network. Defined as $\text{Mish}(x) = x \cdot \tanh(\ln(1 + e^x))$, it is smooth and non-monotonic, enabling better gradient propagation and the retention of small negative values. Studies have shown that Mish improves generalisation and convergence in deep networks, particularly in sign language recognition tasks (Alaftekin et al., 2024). This makes it a strong alternative to ReLU for capturing complex patterns from MediaPipe-extracted keypoints.

Training was conducted using a batch size of `128` and a learning rate of `0.00005`, both selected based on empirical observations. A batch size of `128` offered a good balance between training stability and computational efficiency, enabling smoother gradient estimates without significantly increasing memory requirements. The smaller learning rate allowed for more stable convergence and finer weight adjustments, helping to reduce overfitting and improve generalisation.

6.3.2 Model Architecture

The model was trained using the Adam optimiser on two configurations: one with dense layers of 64, 32 and 16 units (Dense 64/32/16), and another with 64 and 32 units (Dense 64/32), as illustrated in Figure 41.

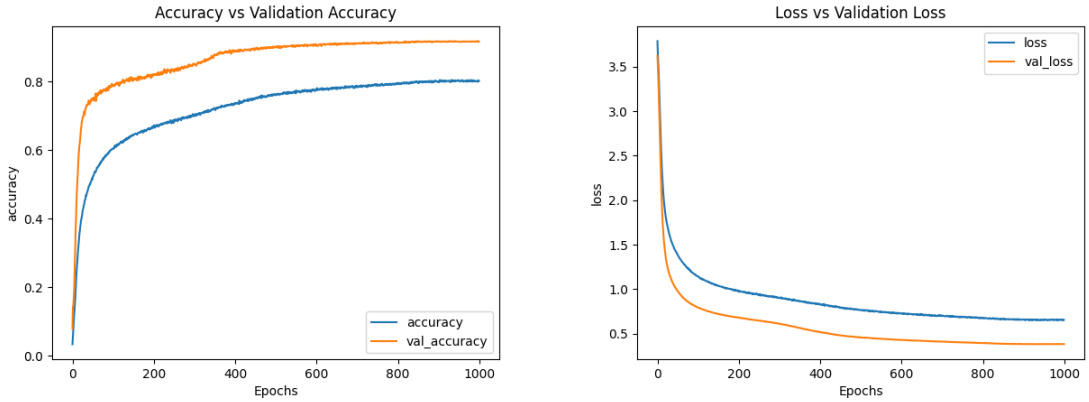


(a) Dense 64/32/16 (b) Dense 64/32

Figure 41: Model summary for Dense 64/32/16 and Dense 64/32

6.3.3 Results

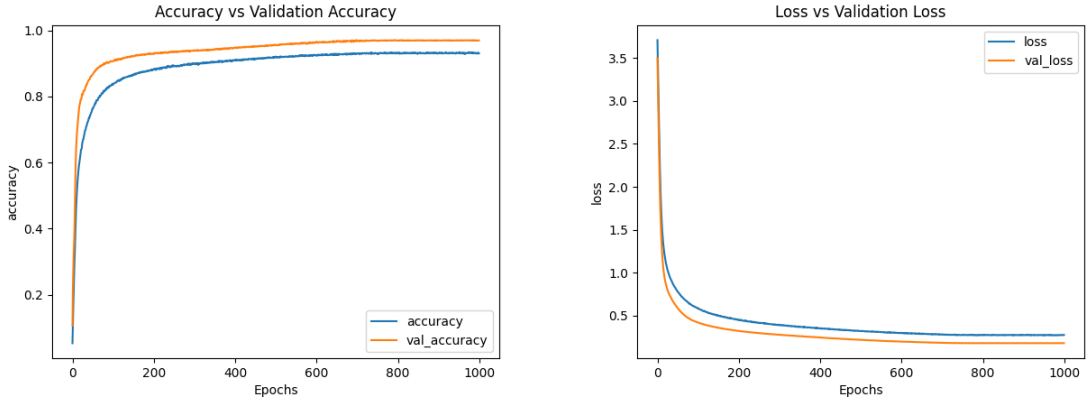
Dense 64/32/16
Figure 42 shows the accuracy and loss curves for the Dense 64/32/16 model over 1000 epochs.



(a) Accuracy (b) Loss

Figure 42: Accuracy and Loss Curves for Dense 64/32/16

Dense 64/32
Figure 43 shows the accuracy and loss curves for the Dense 64/32 model over 1000 epochs.



(a) Accuracy (b) Loss

Figure 43: Accuracy and Loss Curves for Dense 64/32

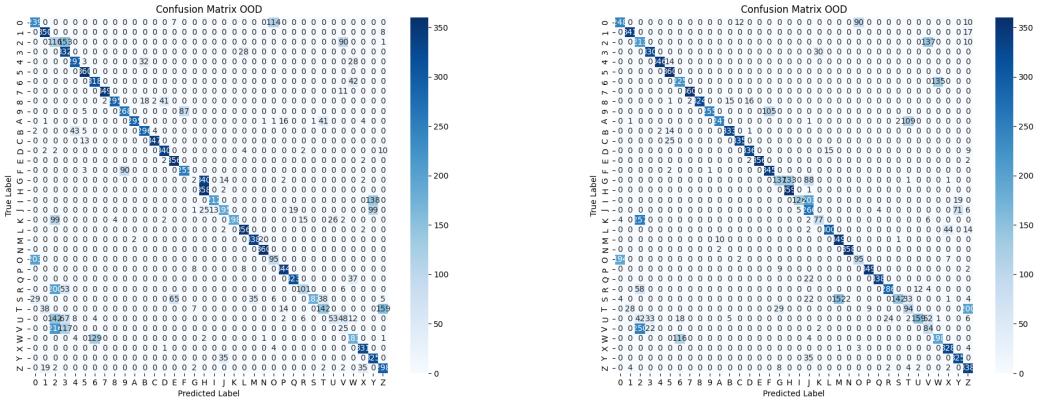
6.3.4 Evaluation

Table 13 presents the best Macro-F1 scores achieved respectively by each model configuration, along with the corresponding epoch at which the score was obtained.

Model Configuration	**Best Macro-F1 Score**	Epoch
Dense 64/32/16	0.7076	750
Dense 64/32	0.7502	800

Table 13: The best Macro-F1 Score based on three runs for each model configuration

Figure 44 presents the confusion matrices for out-of-domain performance across the different best model configurations.



(a) Dense 64/32/16

(b) Dense 64/32

Figure 44: Confusion matrices for out-of-domain evaluation across different best model configurations

The model configuration Dense 64/32 consistently outperformed the deeper Dense 64/32/16 architecture in Macro-F1 (Table 13) score. The noticeable improvements indicate better overall and class-balanced generalisation. However, the relatively tight overlap between training and validation curves suggests that while Dense 64/32 achieves higher performance, it may be more prone to overfitting if training continues unchecked.

Moreover, the Dense 64/32 model exhibited more stable accuracy and lower validation loss during training, suggesting that removing the final Dense 16 layer not only simplified the model but also enhanced its generalisation capabilities.

As shown in Figure 44, the confusion matrix for the Dense 64/32 configuration demonstrates clearer diagonal alignment, reflecting stronger class-wise prediction performance compared to Dense 64/32/16. This further supports the decision to adopt the shallower configuration as the preferred architecture for out-of-domain gesture classification.

To conclude, the FNN model with a Dense 64/32 configuration is preferred over its deeper counterpart, Dense 64/32/16, as it demonstrated more stable training behaviour, and generalised well to out-of-domain data. The results suggest that a simpler architecture is sufficient for this task and may even provide improved performance, though care should be taken to monitor overfitting as the model converges. This reinforces the value of architectural efficiency when working with structured, low-dimensional inputs such as hand keypoints.

6.3.5 Candidate Model

Based on the evaluation results, the FNN model with a Dense 64/32 configuration trained for `800` epochs was selected as the best-performing model. Its confusion matrix, presented in Figure 44b, illustrate its classification performance on the out-of-domain datasets.

7 Chained Model

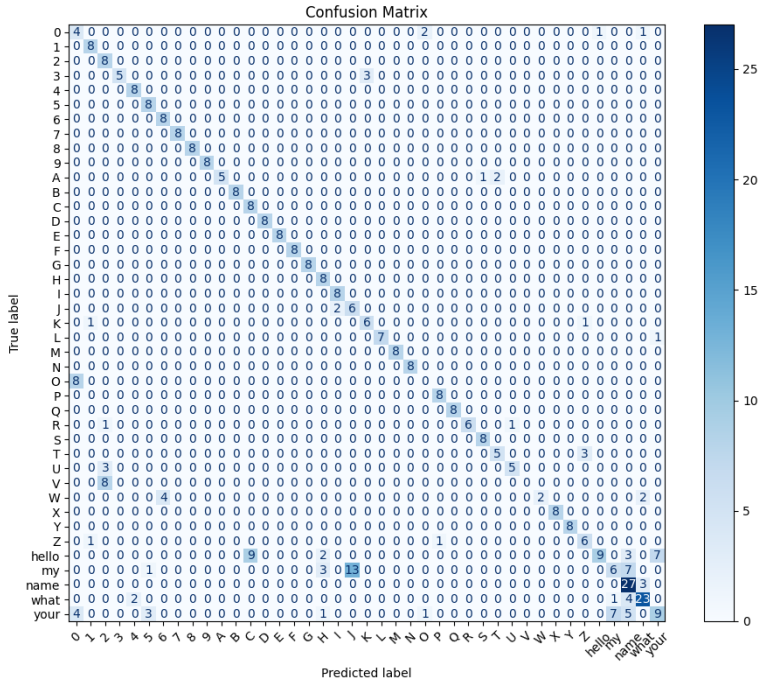


Figure 45: Chained model evaluation on an out-of-domain dataset.

The models were chained sequentially, beginning with the gesture classification model and followed by either the static or dynamic gesture classification model. Figure 45 illustrates the evaluation results of the chained model on an out-of-domain dataset.

The model exhibited difficulty distinguishing between gestures such as *'O'* and *'0'*, *'V'* and *'2'*, and *'W'* and *'6'*. These confusions likely stem from the visual similarity of these gestures in ASL.

For dynamic gestures, the model occasionally misclassified *"my"* as static, possibly due to the relatively small range of motion involved. Additionally, *"hello"* was sometimes predicted as a static gesture, potentially because the motion caused the hand to exit the camera frame, resulting in insufficient sampling of the gesture's movement. However, during live capture, the model tends to classify *"hello"* more accurately, suggesting that real-time motion cues and continuous temporal context may assist in disambiguating dynamic gestures.

7.1 LLM Interpretation

After passing the video feed through the combined model, predictions from the static and dynamic gesture classifiers were concatenated to form a sentence. A LLM was integrated to interpret sequences of ASL gestures into coherent English sentences.

7.1.1 Evaluation

10 participants were tasked to sign two phrases: *"Hello, what is your name?"* and *"My name is . . . "*, with finger-spelled names. To evaluate LLM inference, ASL outputs were passed through grammar-constrained models, each ran three times per sample. BERTScore (Zhang et al., 2019) was used to assess semantic similarity using contextual embeddings, with average F1, Precision, and ?Recall reported. Additionally, model accuracy was computed as the percentage of outputs that exactly matched the ground truth.

Table 14 presents the results of the comparison between Llama3 (nmerkle, 2024), Gemma 1.1 (ggml org, 2024) and TinyLlama (falan42, 2024) in the interpretation of the ASL phrases.

LLM	Average F1	Average Precision	Average Recall
Llama3	0.9018	0.8926	0.9113
Gemma 1.1	0.8745	0.8589	0.8910
TinyLlama	0.8606	0.8694	0.8695

Table 14: The average BERTScore for each metric per LLM

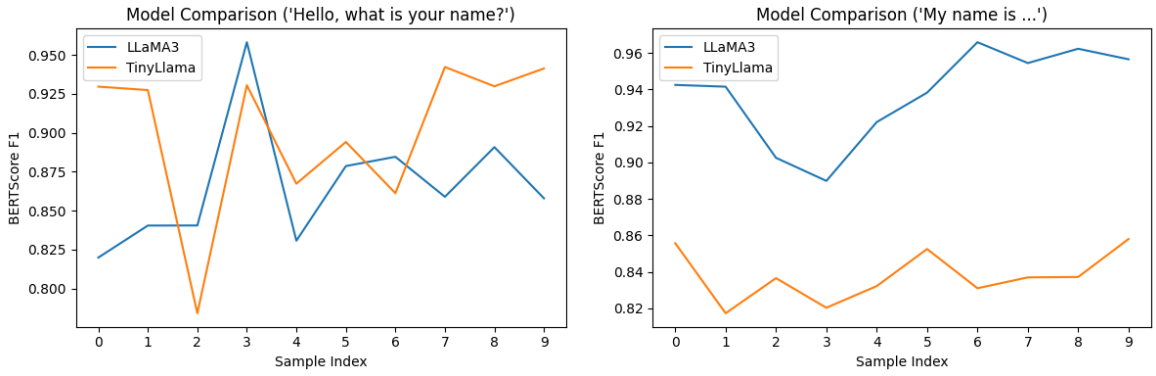
Table 15 presents the respective model sizes, accuracies and average runtimes of Llama3, Gemma 1.1 and TinyLlama.

LLM	Model Size	Accuracy (%)	Average runtime (s)
Llama3	8B	40.0	9.10
Gemma 1.1	7B	15.0	8.28
TinyLlama	1.1B	30.0	33.83

Table 15: Effects of model size on accuracy and average runtime

Llama3 achieved the highest average BERTScore F1 of `0.9018`, outperforming both Gemma 1.1 and TinyLlama. It also outperformed the others in both Precision and Recall, indicating stronger semantic alignment with the ground truth. In terms of exact match accuracy, Llama3 achieved `40.0%`, outperforming TinyLlama's `30.0%`, as seen in Table 15. Despite its smaller size (`1.1B`), TinyLlama was the slowest at `33.83s` due to hallucinations producing overly long outputs. Gemma 1.1 was faster at `8.28s` but less accurate. Llama3, although the largest (`8B`), struck the best balance, achieving the highest accuracy with a moderate runtime of `9.10s` and consistent, grammar-aligned responses.

Figure 46 illustrates the BERTScore F1 performance of Llama3 and TinyLlama across individual ASL samples.



(a) Model comparison for *"Hello, what is your name?"* (b) Model comparison for *"My name is . . . "*

Figure 46: LLM evaluation on an out-of-domain dataset.

As shown in Figure 46, Llama3 excels at both types of sentences, especially those involving finger-spelled names. In contrast, TinyLlama produces a few high-scoring outputs, occasionally matching Llama3. However, its scores fluctuate wildly, reflecting its observed hallucination behavior.

7.1.2 Chosen LLM

Based on the evaluation results, Llama3 demonstrates superior efficiency, consistency, and semantic alignment, making it more suitable for grammar-constrained ASL interpretation. Figure 47 shows an example of the LLM over the user-interface.

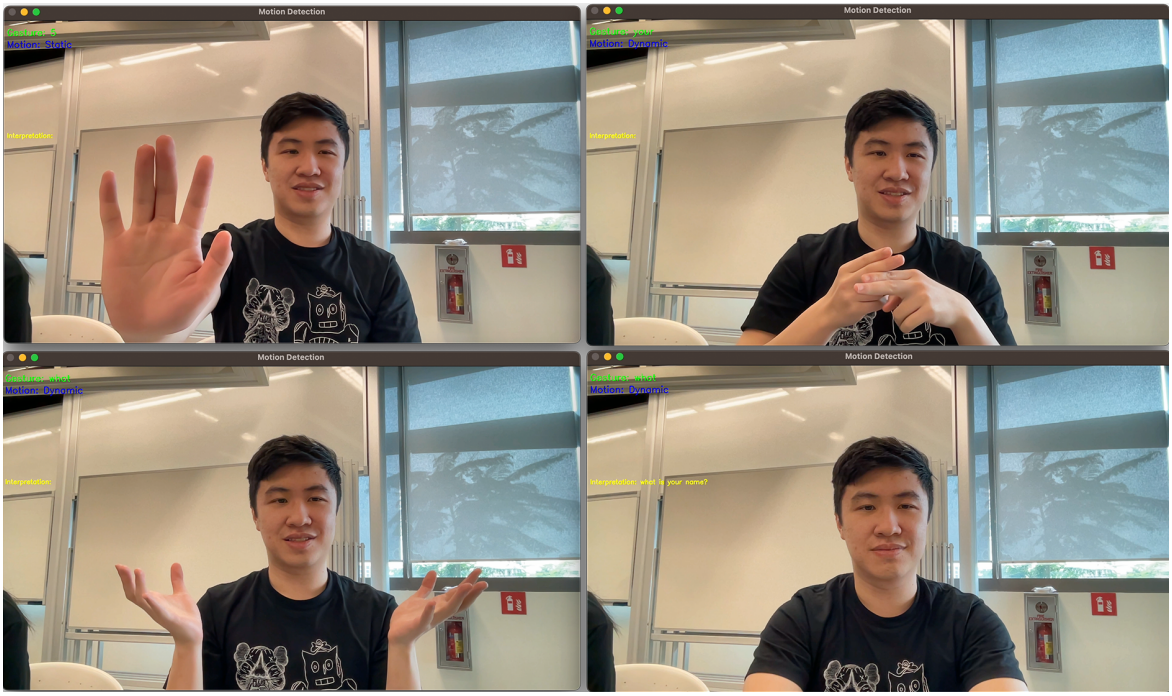


Figure 47: Example user interface of the ASL translation. The model predicts gestures with a slight delay, as each prediction appears one frame after the corresponding gesture is performed.

Conflicts of Interest. The authors declare no conflicts of interest.

Acknowledgment. We would like to thank Professor Poria Soujanya and Professor Dorien Herremans for their guidance in this project.

References

Alaftekin, N., Koç, U., Karabacak, K., and Koçer, S. (2024). Real-time sign language recognition based on yolo algorithm. *ResearchGate*. Accessed: 22 March 2025.

Benhaili, Z., Kabbaj, I., Balouki, Y., and Lahcen, M. (2022). *Human Activity Recognition Using Stacked LSTM*, pages 33–42.

falan42 (2024). tiny_llama-1.1b-v2.gguf. `https://huggingface.co/falan42/tiny_llama-1.1b-v2.gguf`.

ggml org (2024). gemma-1.1-7b-it-q4_k_m-gguf. `https://huggingface.co/ggml-org/gemma-1.1-7b-it-Q4_K_M-GGUF`.

Google (2025). Mediapipe studio: Interactive ml model builder.

Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M., Zhang, F., Chang, C.-L., Yong, M. G., Lee, J., Chang, W.-T., Hua, W., Georg, M., and Grundmann, M. (2019). Mediapipe: A framework for building perception pipelines.

Muhammad, U., Hoque, M. Z., Oussalah, M., and Laaksonen, J. (2023). Deep ensemble learning with frame skipping for face anti-spoofing.

nmerkle (2024). Meta-llama-3-8b-instruct-ggml-model-q4_k_m.gguf. `https://huggingface.co/nmerkle/Meta-Llama-3-8B-Instruct-ggml-model-Q4_K_M.gguf`.

Reddy, B. K. and Anitha, K. (2022). A comparative analysis for measuring accuracy in recognizing hand gestures using feedforward neural network and cnn method. In *2022 International Conference on Business Analytics for Technology and Security (ICBATS)*, pages 1–8.

Ruopp, M. D., Perkins, N. J., Whitcomb, B. W., and Schisterman, E. F. (2008). Youden index and optimal cut-point estimated from observations affected by a lower limit of detection. *Biometrical Journal. Biometrische Zeitschrift*, 50(3):419–430.

Sateesh, S. K., BK, S., and D, U. (2024). Decoding human emotions: Analyzing multi-channel eeg data using lstm networks. *arXiv preprint arXiv:2408.10328*.

Teak-Wei Chong and Boon-Giin Lee (2018). American sign language recognition using leap motion controller with machine learning approach - scientific figure on researchgate. `https://www.researchgate.net/figure/The-26-letters-and-10-digits-of-American-Sign-Language-ASL_fig1_328396430`.

WonderPrint (2025). 11x14 asl phrases poster/asl poster/learn american sign language/home wall decor/home wall art/living room decor/asl dictionary/science poster/sing the alphabets. Accessed: 2025-03-23.

Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., and Artzi, Y. (2019). Bertscore: Evaluating text generation with BERT. *CoRR*, abs/1904.09675.