University of Hertfordshire UH

School of Physics, Engineering and Computer Science

MSc Data Science Project

7PAM2002-0509-2024

Department of Physics, Astronomy and Mathematics

Data Science FINAL PROJECT REPORT

Project Title:

Predictive Modelling of Sign Language Acquisition in Children

GitHub Link:

[Link to Repository]

Student Name and SRN:

Tiasa Malitya 24005709

Supervisor: Carrie Rickets

Date Submitted: 27/08/2025

Word Count: 4962

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science **in Data Science** at the University of Hertfordshire.

I have read the detailed guidance to students on academic integrity, misconduct and plagiarism information at Assessment Offences and Academic Misconduct and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6)

I did not use human participants in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed: Tiasa Malitya

Student Name signature Malitya

Student SRN number: 24005709

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

ACKNOWLEDGMENT

I would like to express my sincere gratitude to my Supervisor Carrie Rickets for her continuous guidance, insightful feedback and encouragement throughout this project. I also acknowledge the researchers who developed and shared the ASL-PLAY dataset which served as the foundation of my analysis. In addition, I am deeply grateful to all of my Professors at the University of Hertfordshire who helped me to gain pure knowledge and build confidence in Data Science. Finally, I am thankful to my peers for their valuable discussions and to my family and friend for their constant support and care during the completion of this work.

ABSTRACT

This study investigates predictive modelling of American Sign Language (ASL) acquisition in children using cognitive, interactional and demographic features from the ASL-PLAY dataset. Motivated by established findings that language development depends on both cognitive skills and environmental input (Tomasello, 1995; Meier, 1991), three models were evaluated: Linear Regression, Random Forest and Long Short-Term Memory (LSTM) networks. Preprocessing involved imputation, feature scaling, outlier handling and for the LSTM sequential grouping of per-child interaction data. Evaluation employed mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE) and R² to assess predictive accuracy.

Results showed that Random Forest outperformed Linear Regression by effectively modelling non-linear feature relationships while LSTM demonstrated strong error minimisation despite lower R², reflecting its sensitivity to limited sequence data. Feature importance analysis identified Child_Age, adult object touch and adult gaze as the most influential predictors aligning with psycholinguistic research on input-driven language acquisition.

These findings highlight the potential for machine learning to inform early ASL intervention strategies and personalised learning tools. Future work will focus on expanding the dataset, incorporating multimodal inputs and exploring advanced sequential models to better capture the temporal observation of sign acquisition.

Table of Contents

Unit I

1. Introduction

This unit explains the background of sign language acquisition in children and why this topic is important. It also shares the motivation behind the data driven approaches for my project and the research questions. Finally, it covers the ethical considerations taken to use the data responsibly for my study.

1.1. The story behind

Sign language acquisition refers to the process through which children learn to use visual-gestural language as a means of communication. Just like spoken language, sign language develops through social interaction, exposure and repeated use. While sign language development shares similarities with spoken language in terms of stages and timing, it remains under-researched especially in terms of large-scale data analysis and measurable developmental patterns (Meier, 1991).
My personal interest in this area began with my uncle, who is deaf and grew up using sign language. It helped me see how important early language support is for deaf children to develop.

Past work on sign language acquisition has largely relied on theoretical analysis (see Appendix A). My project shows a predictive modelling turn to this problem. By applying regression models and Long Short-Term Memory (LSTM) networks, I aim to explore how different features like age, eye gaze and interactive behaviours relate to a child's ASL learning progress that is ASL-Score.

1.2. Motivation

Both personal and academic reasons motivated me to choose this project topic. It led me to wonder how children learn to sign and what factors help or hinder.
From an academic perspective, I wanted to explore how data science can support child development research. Sign language acquisition is a difficult process which is depend on many features, including age, interaction patterns, attention and family background. I believed that using machine learning models could help uncover which of these features play the biggest role in a child's sign learning progress.

1.3. Objective

This project is designed around three key objectives:

Predictive Modelling
To build and compare predictive models that estimate a child's ASL learning progress. This includes applying regression models such as linear regression and random forest as well as time-series models like LSTM.

Feature Exploration
To examine which developmental and interactional features including age, gaze and parental hearing status play the most significant role in predicting ASL acquisition.

Evaluation, Optimisation and Application
To evaluate, optimise the models and interpret the results in terms of real-world applicability, with the aim of supporting early intervention strategies for children learning ASL.

1.4. Research Topic and Question

Most existing research in sign language and machine learning has focused on recognition tasks like identifying signs from images or videos. In contrast, I have tried to explore Predictive Modelling of sign language acquisition in children: how children learn ASL over time and whether that learning progress can be predicted using behavioural and demographic features.
Based on the topic, the guiding research question is:

Which model better predicts Sign Language acquisition progress in children using cognitive and environmental features — Regression or LSTM?

Since this project is exploratory and the models show moderate performance, the research question is approached as an open investigation. The aim is more likely not to prove which model is best but to examine whether such models can work with this data, the patterns they reveal and the challenges involved.

1.5. Ethical Consideration

This project uses data collected from children interacting with adults in a natural play environment. Since the data involves minors and sensitive behavioural observations ethical handling is important. The following points address the key ethical and legal considerations as required by the project guidelines:

Q1. Does the data meet GDPR requirements?
Yes. The dataset is fully anonymised. It does not contain any names, identifiable faces, addresses or sensitive personal information. It meets GDPR requirements for data privacy and protection.

Q2. Does this project meet the UH Ethics Policy for research?
Yes. The project is designed following the University of Hertfordshire's research ethics policy. Only secondary data has been used with no direct involvement of human subjects and adheres to guidelines for safe, responsible and ethical use of data involving minors.

Q3. Do you have permission to use the data for research?
Yes. The ASL-PLAY dataset is publicly available for academic research and is hosted on the Open Science Framework (OSF). It is shared under a Creative Commons Attribution 4.0 International License (CC BY 4.0) which allows reuse with attribution. No payment or special permission is required to access or use the dataset.
Reference: Lieberman, A., Gappmayr, P. & Fitch, A. (2022) *ASL-PLAY: A dataset of early American Sign Language acquisition* [dataset]. Open Science Framework. Available at: https://osf.io/3w8ka/ (Accessed: 9 July 2025).

Q4. Was the data ethically collected?
Yes. The data were collected in a controlled setting to study child language development, with video recordings conducted under parental consent. Trained researchers later coded the sessions and the original authors shared the dataset on OSF. The authors have confirmed that ethical procedures were properly followed and that they have the right to share the data.

All data used in this project was downloaded directly from the OSF repository and stored securely on university-approved platforms. It is used strictly for academic purposes within the scope of this MSc project and no data will be redistributed.

Unit II

2. Literature Review

In this unit, I mentioned previous research on sign language acquisition and the use of machine learning in language studies. I highlighted key methods from the literature including traditional statistical models and deep learning approaches.
The review begins with an overview of selected studies, followed by a comparison between the models used in those studies and the models implemented in this project.

2.1. The choice of Literature

The literature reviewed for this project includes studies from three main areas: sign language acquisition in children, machine learning techniques applied to language development, and model-based prediction in developmental research. These studies were selected based on their relevance to the project's focus on modelling sign language learning, particularly in early childhood contexts.

As I previously mentioned that sign language research has centred around visual recognition where computer vision techniques are used to identify signs from video input (Koller et al., 2019; Camgoz et al., 2018). However, fewer studies explore **how children acquire sign language over time**, and even fewer attempt to **predict** learning progress using behavioural features. This tells the reason behind my interest to explore that side.

In addition, work by Mayberry and Lock (2003) emphasises the importance of **age of first language exposure**, showing that earlier access to language (signed or spoken) leads to stronger long-term linguistic outcomes. This is particularly relevant to this project, as age is one of the key features explored in the modelling process. The review also includes research on **machine learning models** in developmental contexts, such as regression models used to predict learning outcomes (Lipnevich et al., 2016) and LSTM models applied in sequential learning tasks (Greff et al., 2017). These papers informed the decision to test both regression and LSTM approaches in this project.

The selected literature therefore serves two purposes: it provides a foundation for understanding existing methods in sign language and child development research and it helps justify the choice of models implemented in this project.

2.2. Comparison with Literature

To compare this project's approach with existing work, TABLE-I summarises key studies. These papers were selected for their relevance to the project's objectives: modelling ASL acquisition in children using cognitive and environmental features. The table outlines each study's focus, methods used and how they relate to the models and evaluation strategies employed in this project. This structured comparison highlights both the theoretical foundations and methodological contrasts that guided the model choices.

Author	Study Focus	Method Used	Dataset	Insights	Relevance to my Project
Orlansky & Bonvillian (1985)	Language development in children of Deaf parents	Observational research	Deaf children of Deaf vs. hearing parents	Rich interaction leads to better ASL acquisition	Supports inclusion of interaction and parental hearing variables
Greff et al. (2017)	Evaluating LSTM models for time-series data	LSTM analysis across tasks	Various time series datasets	LSTM excels in handling sequential data	Justifies use of LSTM for modelling ASL-Score progress over timing of
Camgoz et al. (2018)	Sign language translation using computer vision	CNN + RNN for sign recognition	RWTH-PHOENIX-Weather 2014T dataset	High performance in visual sign recognition	Shows ML use in sign language
Lipnevich et al. (2016)	Predicting academic success from behavioural traits	Linear regression, statistical analysis	Student behavioural survey data	Emotion and behaviour predict learning outcomes	Justifies use of **regression models** for predicting child development scores
Mayberry & Lock (2003)	Investigating age of first language exposure on ASL proficiency	Developmental psycholinguistic analysis	Data from Deaf individuals with varied age of ASL exposure	Early exposure leads to better language outcomes	Supports use of **age** as an important predictive feature

TABLE-I [Appendix B]

Unit III

3. Dataset

This unit outlines the dataset used in the project, including its source, structure and key characteristics. It also describes the steps taken to clean, filter and prepare the data for both regression and time-series modelling.

3.1. Description

The dataset for this project comes from the **ASL-PLAY study** on American Sign Language (ASL) acquisition and visual attention in Deaf and hard-of-hearing children (Lieberman, Gappmayr, & Fitch, 2022). It contains video-coded **parent–child play sessions** collected across multiple U.S. sites. For this project, I used only **Session 1 (Play1.0)**, which includes data from **24 children** aged **9–60 months** (mean = 37 months) because Session 2 (Play2.0) used a different coding scheme that reduced comparability. [Detailed in Appendix C]

3.2. *Why ASL-Play Dataset*

I chose the ASL-PLAY dataset because it goes beyond sign recognition and captures the developmental process of language acquisition. It combines behavioural features such as gaze, object use and ASL signs with demographic factors like age and parental hearing status (Lieberman, Hatrak & Mayberry, 2014). This structure enabled me to feature engineer the **ASL_Score** variable as the target which made the dataset suitable for applying both regression and time-series approaches.

3.3. *Steps in Preprocessing*

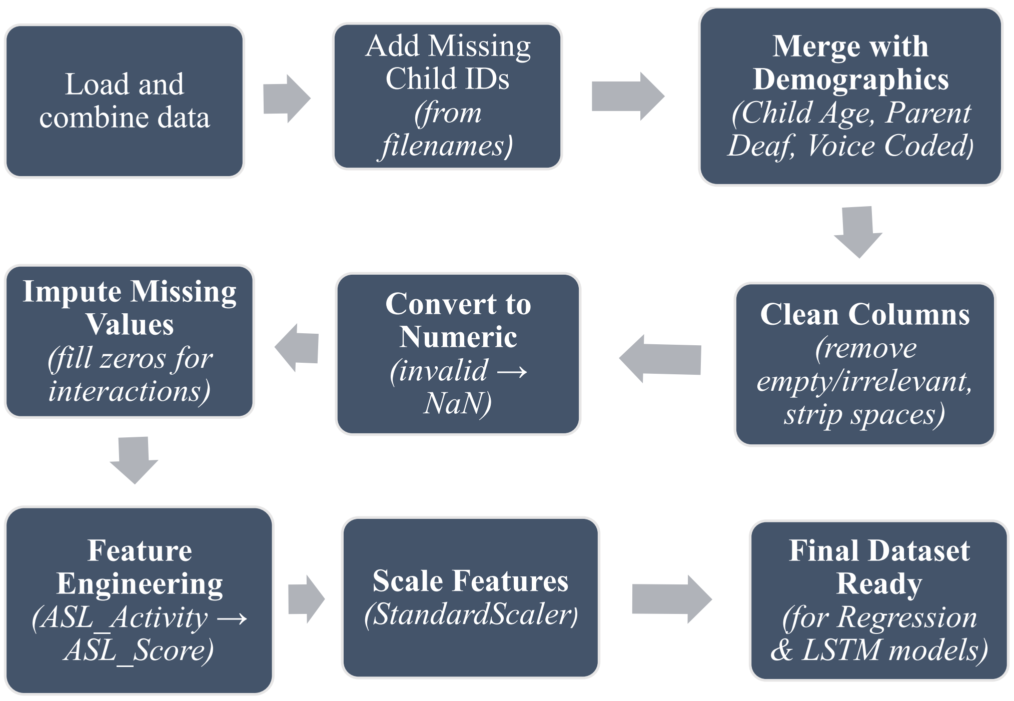

Load and combine data
Add Missing Child IDs (from filenames)
Merge with Demographics (Child Age, Parent Deaf, Voice Coded)
Clean Columns (remove empty/irrelevant, strip spaces)
Convert to Numeric (invalid → NaN)
Impute Missing Values (fill zeros for interactions)
Feature Engineering (ASL_Activity → ASL_Score)
Scale Features (StandardScaler)
Final Dataset Ready (for Regression & LSTM models)

FIG - 1[Appendix D]

To prepare the dataset, I combined 24 child interaction files and aligned them with demographic data using Child ID. After cleaning and handling missing values, I engineered two key features: [detailed justification in Appendix E] **ASL_Activity** at the interaction level and **ASL_Score** at the child level, which became the target for my models. Finally, I scaled the numerical variables to ensure consistency across features before training regression and time-series models.

3.4. *Exploratory Data Analysis*

Before applying any model, I have explored the data to better understand the distribution, patterns and data quality. This involved investigating the target variable (ASL_Score) and its relationship with key features such as Child_Age and Parent_Deaf status and guide the selection of features for predictive modelling. I used histogram, boxplot, scatterplot and bar plot visualisations to explore distribution, outliers and variable relationships relevant to the problem of predicting sign language acquisition in children.

Histogram:

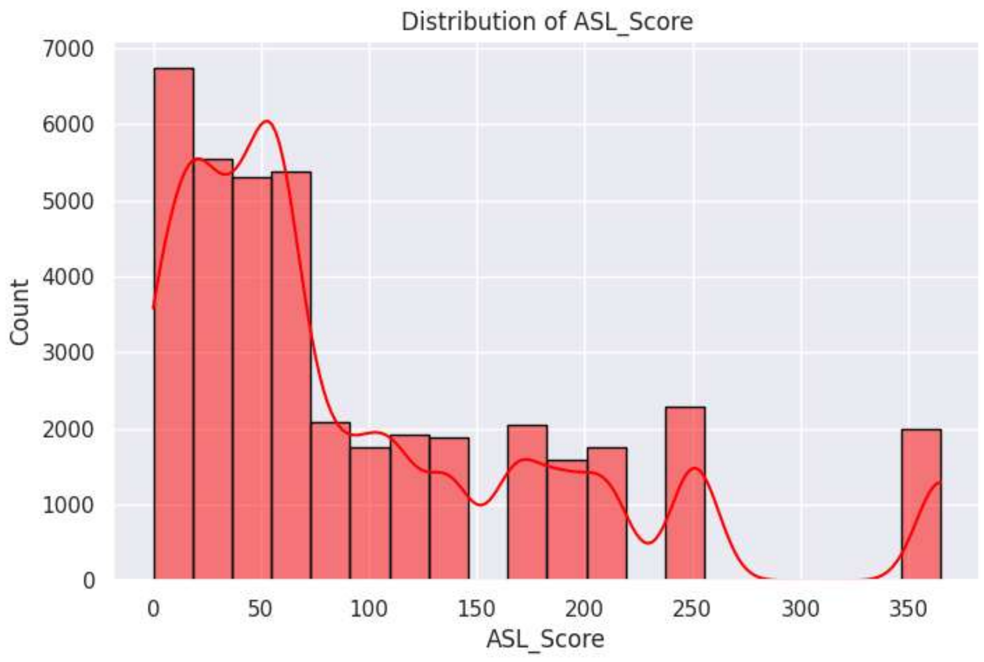


FIG - 2[Appendix F]

The histogram showed that most children had low ASL_Score values, mostly between 0 and 50, indicating limited use of sign language during the sessions. A few children had much higher scores, but they were rare, resulting in a skewed distribution. This unbalanced spread suggests that prediction could be challenging for some models and highlights why including features like age and parental hearing status could be important.

Boxplot:

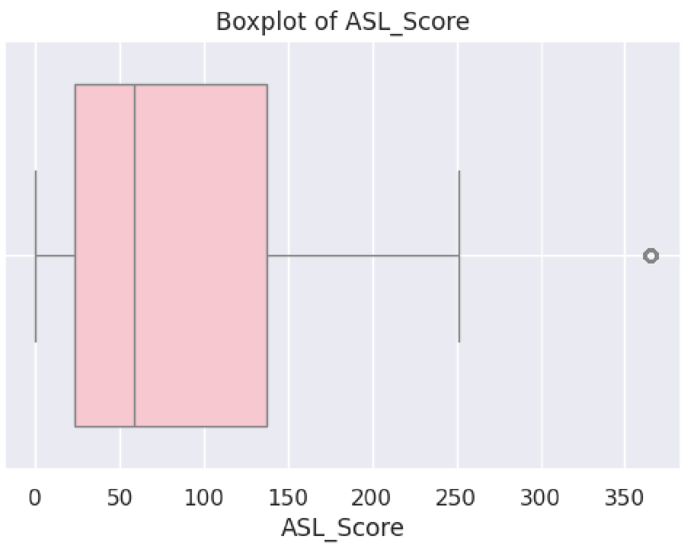


FIG - 3[Appendix G]

The boxplot confirmed that most scores were clustered near the lower end, with low variability among most children. However, one extreme outlier near 350 stood out, representing an unusually high sign language use or a unique case. This reminded me to be cautious when averaging or normalising the data, as extreme values could affect model performance.

Scatterplot (Age(month) vs ASL_Score):

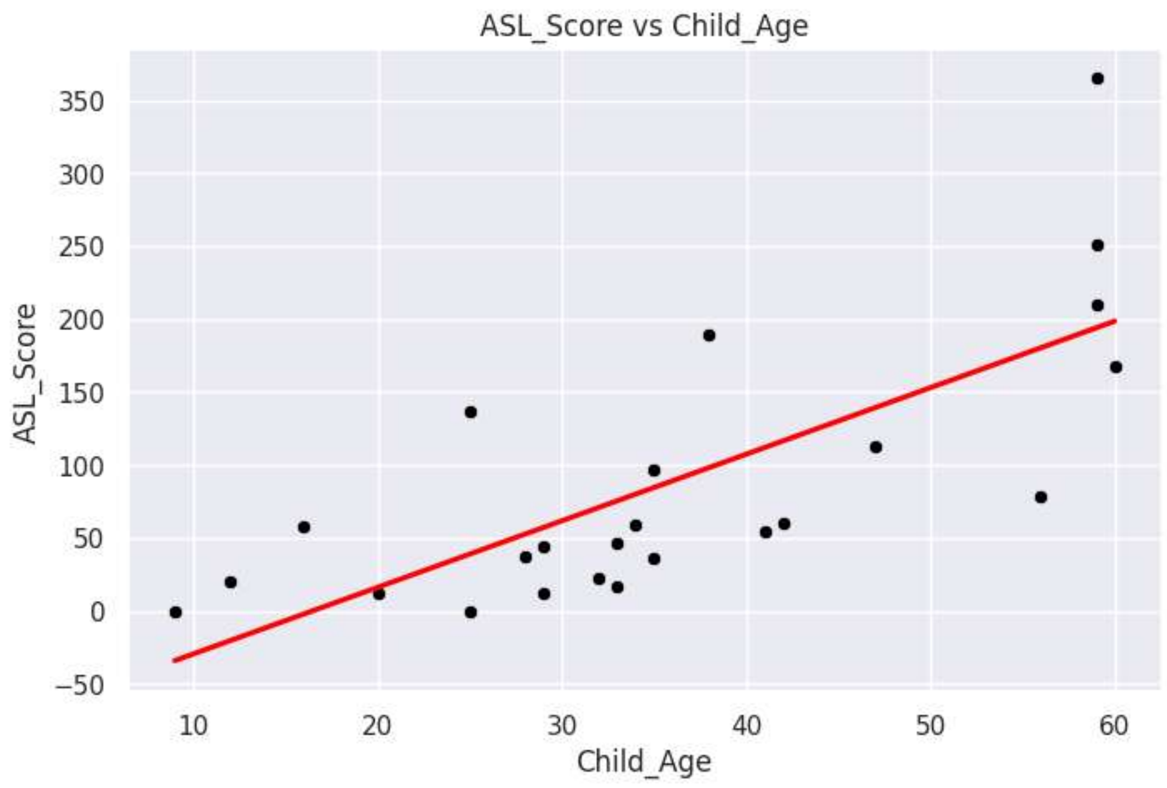


FIG - 4[Appendix H]

The scatterplot revealed a general positive trend between age and ASL_Score: older children tended to use more signs. Yet, there was considerable variation at each age, indicating that age alone is not a strong predictor. This confirms that while age is important, other factors also influence ASL acquisition.

Bar Chart (Parent Deaf Status):

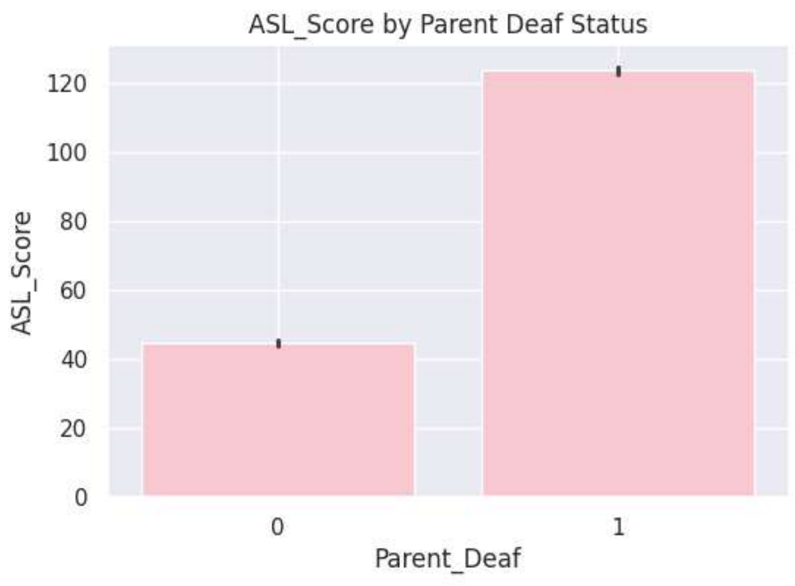


FIG - 5[Appendix I]

The bar chart showed a clear difference in scores between children with deaf and hearing parents, with deaf-parent children scoring about 75 points higher on average. This emphasizes the significance of the home environment in shaping language development and confirms that including parental hearing status is essential for accurate modelling.

Unit IV

4. Methodology

After understanding the dataset, this section outlines the modelling approach taken to predict sign language acquisition. The modelling choices were confirmed by both the structure of the dataset and relevant literature.

4.1. Regression Model

To explore the relationship between cognitive and environmental features and ASL acquisition progress, I first applied regression-based approaches. Two regression models were implemented: **Linear Regression** and **Random Forest Regressor**.

Implementation of Linear Regression

I have picked Linear Regression as it shows simplicity, interpretability and ability to reveal direct positive relationships between features and the target variable (ASL_Score). This approach is commonly used in developmental and behavioural studies where additive, interpretable effects are important (Gelman & Hill, 2007; Cohen et al., 2003). Standardising the data with StandardScaler ensured that all features contributed equally, while allowing me to assess the relative influence of each predictor.

Implementation of Random Forest

Secondly, I have tried with Random Forest as there is a chance of complex dependencies in the interaction data. This model constructs several decision trees and combines their outputs by averaging the predictions, effectively capturing interactions and non-linear patterns (Breiman, 2001). It also provides feature importance measures aiding interpretability. As Random Forest doesn't react to feature scaling, this model complemented the linear baseline by offering a more flexible approach.

Both models used a consistent train-test split, initially set at 20% for testing. With only 24 children, this small test set led to poor generalisation likely due to high variance in performance metrics and unreliable evaluation (Kohavi, 1995). Increasing the test size to 40% provided a better balance between training and evaluation, allowing more reliable assessment of model generalisation and comparison between linear and non-linear approaches.

4.2. Time-Series Model

To capture the sequential nature of children's interactions over time, I implemented an LSTM (Long Short-Term Memory) model, a type of Recurrent Neural Network (RNN) good for time-series data because it can retain temporal relations through memory cells and gating mechanisms (Hochreiter &

Schmidhuber, 1997; Lipton et al., 2015). This makes LSTMs appropriate for modelling behavioural patterns across multiple interaction steps per child.

Since the raw data consisted of interaction-level rows, I first selected relevant interaction and demographic features, including eye gaze, object touch behaviours, child age, parental hearing status, and voice usage. All feature columns were converted to numeric values and missing entries were imputed with zeros to ensure no gaps would bias the training process. I then standardised the features using StandardScaler to normalise their ranges, which stabilised the model training.

Next, I grouped the dataset by Child ID to create one sequence per child, retaining only sequences with at least 10-time steps to provide sufficient temporal information for learning. The target variable for each sequence was computed as the average ASL activity across all interactions of that child. Finally, sequences were padded with zeros to ensure a uniform length, making them compatible with the LSTM input requirements.

The model architecture has been implemented using Keras' Sequential API which contains the following layers:

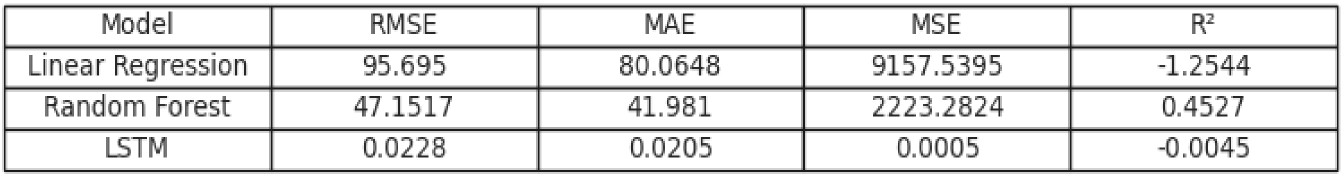
- **Input layer:** Receives padded sequences of 9 features per time step (eye gaze, object touch, age, parental hearing status, voice use). Padding ensures uniform sequence length.
- **LSTM layer (64 units):** Learns time-based dependencies from sequential data. With return_sequences=False, it outputs just the final hidden state, summarizing the entire input sequence.
- **Dense layer (32 units, ReLU** [Appendix J]): Learns higher-level patterns from the LSTM output, adding non-linearity.
- **Dropout layer (0.3**): Helps prevent overfitting by randomly turning off 30% of neurons during training, encouraging the model to generalize rather than depend on specific features..
- **Output layer (1 unit):** Predicts the average ASL activity per child sequence, summarising their sign language use**.**

The model was trained with the Adam optimizer for 30 epochs and a batch size of 8, appropriate for the small dataset. A custom learning rate scheduler reduced the learning rate exponentially after 10 epochs to improve convergence. The dataset was split into 80:20 train-test sets, reserving 20% for evaluation. Unlike the regression models that worked at the aggregate level, this LSTM model uses sequence-level data to capture temporal patterns, making it particularly suited to modelling the dynamics of ASL acquisition over multiple interactions.

Unit V

5. Model Evaluation and Analysis

To assess how well each model predicted ASL acquisition scores, I evaluated them using standard regression metrics: **Mean Squared Error (MSE)**, **Mean Absolute Error (MAE)**, and **R-squared (R^2)**. I have selected these metrics to capture different aspects of model performance. MSE penalises larger errors more efficiently, MAE gives an average deviation, and R^2 indicates the proportion of variance produced by the model.



Model	RMSE	MAE	MSE	R²
Linear Regression	95.695	80.0648	9157.5395	-1.2544
Random Forest	47.1517	41.981	2223.2824	0.4527
LSTM	0.0228	0.0205	0.0005	-0.0045

TABLE-II [Appendix K]

Linear Regression

Linear Regression struggled to model ASL acquisition, as reflected by its high RMSE (95.70) and MAE (80.06), along with a strongly negative R^2 (-1.25). These metrics indicate the model consistently failed to get the underlying structure in the data. The poor performance can be attributed not only to the inherent non-linear nature of language development but also to the limited sample size, which amplifies sensitivity to outliers and sparse features (Tomasello, 2003). While simple regression provides a useful baseline, it lacks the flexibility to model interactions between behavioural cues and demographic variables, which are critical in predicting ASL progression. This highlights the importance of using models capable of capturing complex dependencies in developmental datasets.

Random Forest

The Random Forest Regressor showed substantially better performance, with lower errors (RMSE = 47.15, MAE = 41.98) and a positive R^2 of 0.45. These metrics reflects that the model explained nearly half of the variance in ASL scores, demonstrating its ability to capture non-linear and interaction effects between features. For example, the model can account for how child's age interacts with attention-related behaviours such as gaze and object touch, reflecting the multimodal nature of early language acquisition observed in previous studies (Meier, 1991). Despite this improvement, the model still left a significant portion of variance unexplained, likely due to the small dataset and individual variability among children, illustrating the limitations of classical ensemble methods when applied to sparse behavioural data.

LSTM

The LSTM model achieved extremely low absolute errors (MAE $\approx$ 0.02, RMSE $\approx$ 0.02), suggesting it closely fit the observed sequences. However, the near-zero R^2 (-0.0045) indicates the model did not meaningfully generalise across the group. This discrepancy is largely due to the dataset's small size and irregularity, which limited the LSTM's ability to exploit its temporal modelling advantages. LSTMs are designed to leverage long sequences for capturing dynamic patterns, but with only 24 sequences of variable lengths, generalisation is constrained (Hochreiter & Schmidhuber, 1997). Nevertheless, the low error shows that LSTM can detect subtle patterns in individual sequences.

Unit VI

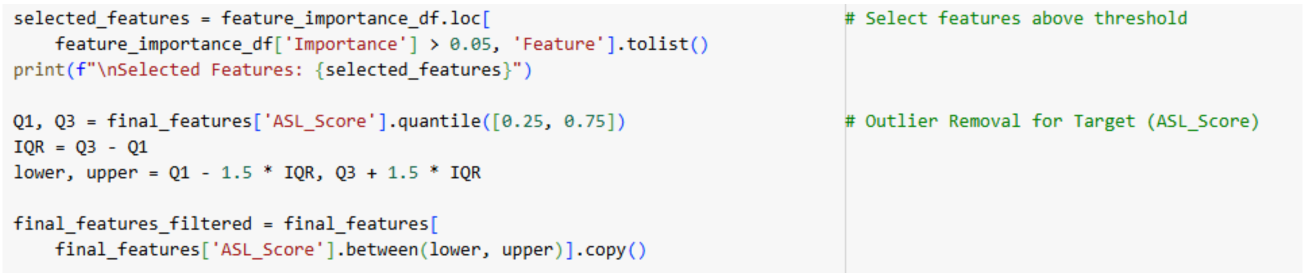
6. Optimisation

To enhance model performance and ensure robustness, several optimisation strategies were employed across the models. These included techniques such as feature importance, careful tuning of model hyperparameters and the integration of a dynamic learning rate schedule in the LSTM model. The goal was to reduce overfitting, improve generalisation on unseen data and tailor each model to the unique characteristics of the ASL-PLAY dataset. This section outlines the specific optimisation efforts applied to each model and discusses how these adjustments contributed to performance improvements or struggles.

6.1. Techniques to optimise

Linear Regression

To improve the robustness of Linear Regression, which is highly sensitive to irrelevant predictors and outliers, several optimisation steps were introduced:



```
selected_features = feature_importance_df.loc[                        # Select features above threshold
    feature_importance_df['Importance'] > 0.05, 'Feature'].tolist()
print(f"\nSelected Features: {selected_features}")

Q1, Q3 = final_features['ASL_Score'].quantile([0.25, 0.75])            # Outlier Removal for Target (ASL_Score)
IQR = Q3 - Q1
lower, upper = Q1 - 1.5 * IQR, Q3 + 1.5 * IQR

final_features_filtered = final_features[
    final_features['ASL_Score'].between(lower, upper)].copy()
```

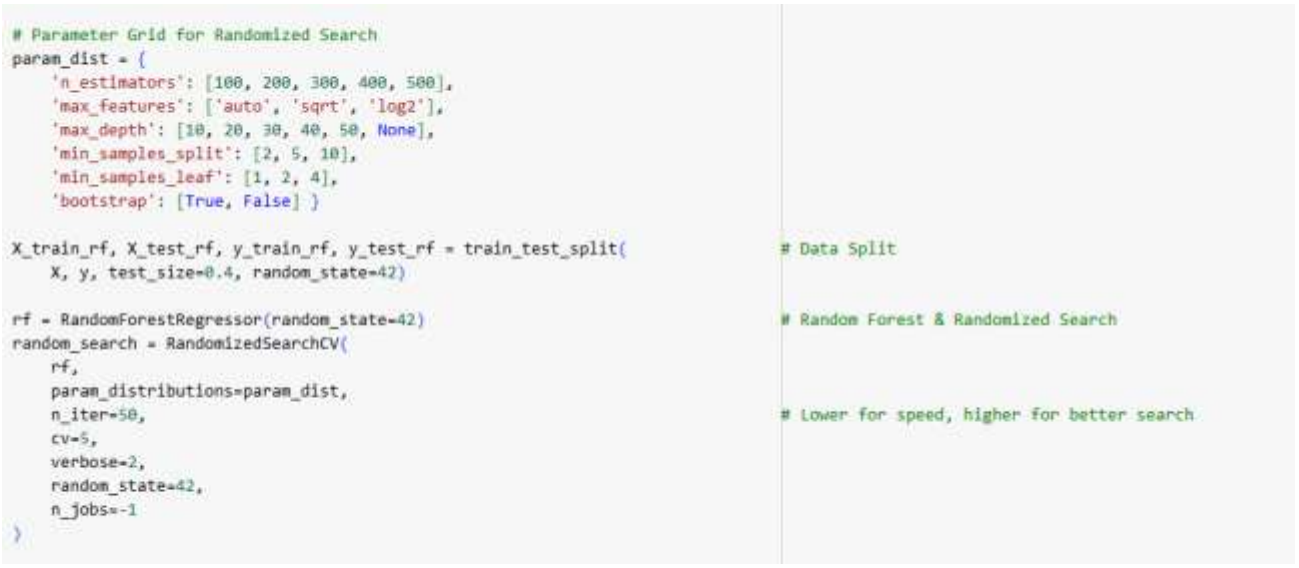
[Detailed explanation in Appendix L]

- **Outlier Handling**: ASL_Score outliers were identified through the Interquartile Range (IQR) method. One extreme observation was removed, ensuring that the regression line was not disproportionately influenced by anomalous values.
- **Feature Selection**: Random Forest feature importances [Appendix M] were used to identify and retain only the most influential predictors (e.g., *Child_Age*, *Object touch (adult) Left*). This reduced dimensionality.
- **Feature Scaling**: StandardScaler was applied to normalise predictor values, ensuring that all features contributed comparably to the model coefficients.

These refinements were designed to stabilise the regression process, enhance interpretability and reduce error propagation from noisy or weak predictors.

Random Forest

The Random Forest model was optimised primarily through **hyperparameter tuning** to enhance its generalisation ability:



```
# Parameter Grid for Randomized Search
param_dist = {
    'n_estimators': [100, 200, 300, 400, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [10, 20, 30, 40, 50, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False] }

X_train_rf, X_test_rf, y_train_rf, y_test_rf = train_test_split(     # Data Split
    X, y, test_size=0.4, random_state=42)

rf = RandomForestRegressor(random_state=42)                          # Random Forest & Randomized Search
random_search = RandomizedSearchCV(
    rf,
    param_distributions=param_dist,
    n_iter=50,                                                       # Lower for speed, higher for better search
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=-1
)
```

[Detailed explanation in Appendix N]

- **Hyperparameter Search with RandomizedSearchCV**:
  Used RandomizedSearchCV to efficiently test combinations of tree depth, number of trees, and node constraints.

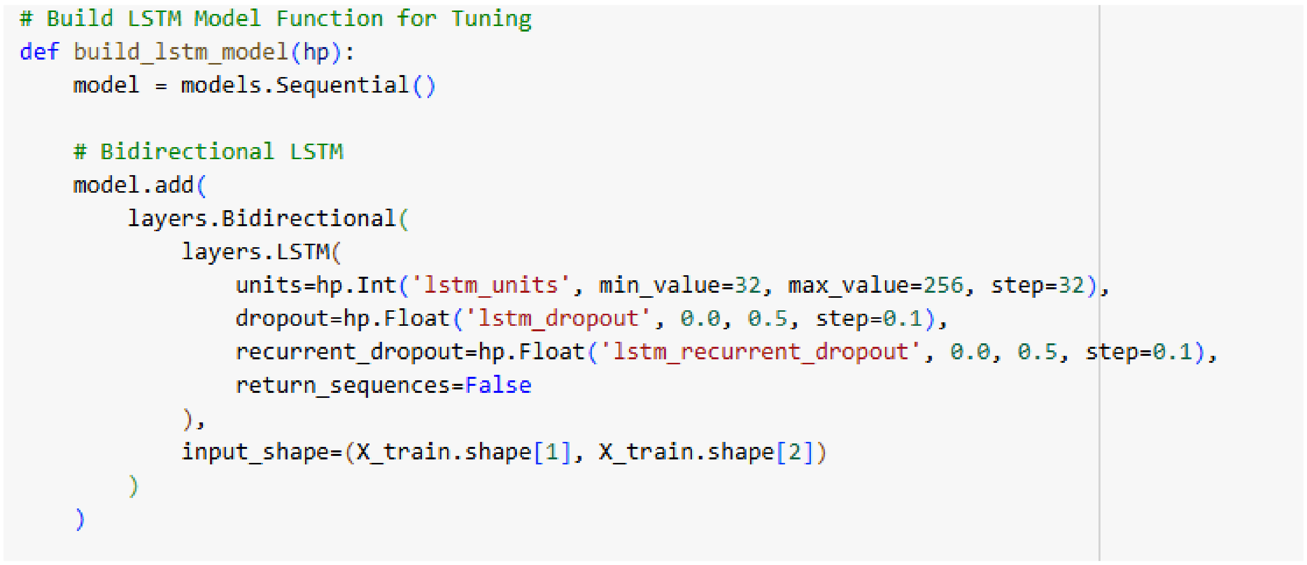
- **Avoiding Overfitting**:
  Limited depth and adjusted leaf sizes to reduce noise memorisation.

- **Use of Full Feature Set**:
  Retained full feature set, as Random Forests are robust to irrelevant variables.

These optimization strategies aimed to balance bias and variance, enabling the model to learn children's ASL acquisition data effectively while avoiding overfitting due to the limited dataset size.

LSTM

The LSTM model was optimised through a combination of architectural tuning and training regularisation. A **Bidirectional LSTM** was explored to understand temporal dependencies in both directions, while hyperparameter tuning explored the number of units, dropout rates, dense layer size and learning rate.



```python
# Build LSTM Model Function for Tuning
def build_lstm_model(hp):
    model = models.Sequential()

    # Bidirectional LSTM
    model.add(
        layers.Bidirectional(
            layers.LSTM(
                units=hp.Int('lstm_units', min_value=32, max_value=256, step=32),
                dropout=hp.Float('lstm_dropout', 0.0, 0.5, step=0.1),
                recurrent_dropout=hp.Float('lstm_recurrent_dropout', 0.0, 0.5, step=0.1),
                return_sequences=False
            ),
            input_shape=(X_train.shape[1], X_train.shape[2])
        )
    )
```

[Detailed Explanation in Appendix O]

- **Network Architecture Adjustments**:
  The number of LSTM units and hidden dense units were tuned to avoid overfitting on a small dataset. Too many units can memorise noise, while fewer units encourage the model to learn only the most relevant sequential patterns.

- **Regularisation Techniques**:
  Dropout and recurrent dropout was applied to reduce reliance on specific neurons, improving generalisation. EarlyStopping was used to stop training once validation loss plateaued, preventing unnecessary overfitting. A learning rate search was also included, since the step size strongly influences stability and convergence speed in sequence models.

- **Data Handling**:
  Padding ensured that child sequences of different lengths could be compared within the same batch, while scaling helped stabilise training by keeping input values within a similar range.

- **Hyperparameter Optimisation:**
  Random search was used to identify an optimal configuration across LSTM units, dropout rates, and learning rates, providing a more systematic approach to tuning compared to manual trial-and-error.

Overall, these strategies were aimed at balancing model complexity with dataset size, ensuring the LSTM learned meaningful temporal trends in ASL activity without overfitting.

6.2. Improvements After Optimisation

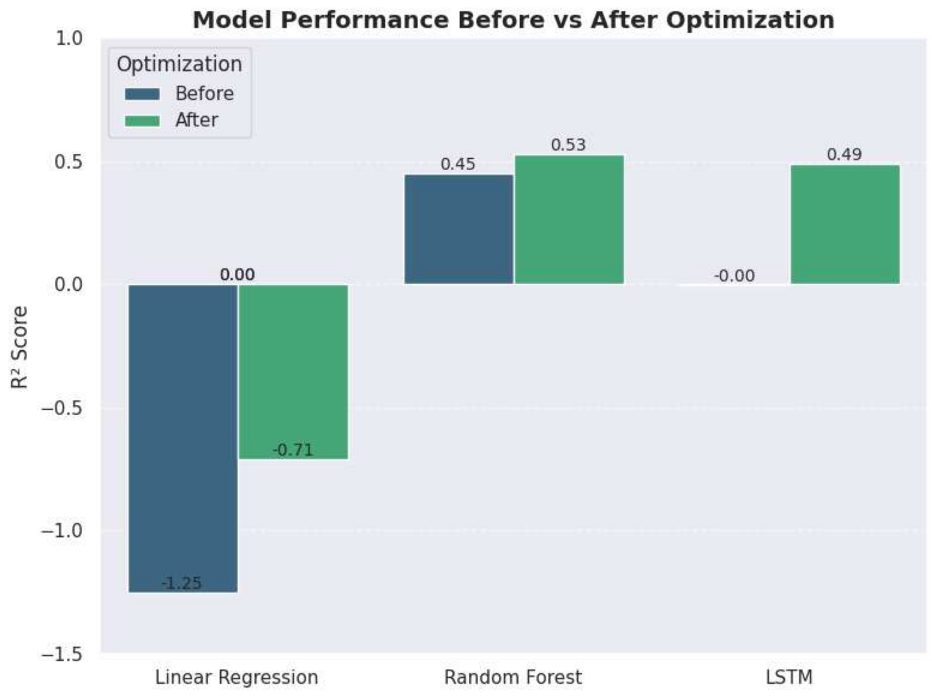
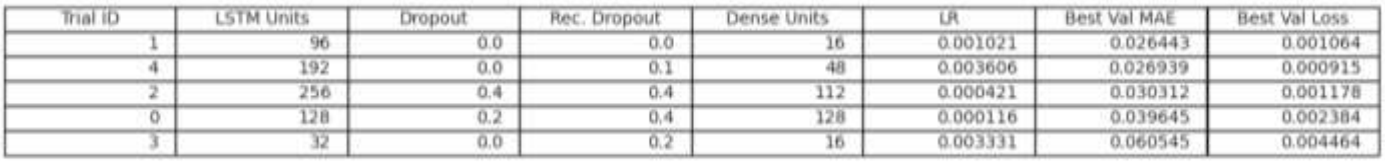


FIG - 6[Appendix P]

Linear Regression
After optimisation, Linear Regression showed modest improvements (MAE: 63.18, RMSE: 73.97, R^2: -0.71). The main gains came from removing an extreme outlier and selecting the most relevant features (Child_Age, Object touch – adult left/right, EYE GAZE – adult), which reduced noise and made the model more stable. Despite this, the model still struggled because it assumes linear relationships, context-dependent relationships in ASL acquisition where multiple behavioural and environmental factors interact (Bates & Goodman, 1997; Kuhl, 2004). This explains why the R^2 remained negative, showing this model could not fully gather the variability in children's ASL scores.

Random Forest
Random Forest improved after tuning hyperparameters, such as the number of trees, tree depth, and minimum samples per leaf. This reduced the MAE to ~38 and RMSE to ~44, while R^2 increased to 0.53. The improvement shows that careful parameter tuning can balance overfitting and underfitting, advancing the model to capture complicate interactions between features like age, eye gaze and object touch behaviours. These results highlight that non-linear models like Random Forest are better suited for small, multimodal datasets, consistent with research emphasizing the role of multimodal cues in early language acquisition (Baldwin, 1995).

LSTM



Trial ID	LSTM Units	Dropout	Rec. Dropout	Dense Units	LR	Best Val MAE	Best Val Loss
1	96	0.0	0.0	16	0.001021	0.026443	0.001064
4	192	0.0	0.1	48	0.003606	0.026939	0.000915
2	256	0.4	0.4	112	0.000421	0.030312	0.001178
0	128	0.2	0.4	128	0.000116	0.039645	0.002384
3	32	0.0	0.2	16	0.003331	0.060545	0.004464

TABLE-III [Appendix Q]

The LSTM became more stable and achieved lower errors (MAE ~0.01, RMSE ~0.02, R^2 ~0.49) However, due to the small dataset and the stochastic nature of training (random weight initialisation and batch sampling), the R^2 value varied slightly across runs. This variability reflects sensitivity to data size rather than flaws in the model design. Optimisation helped the model learn patterns over time in children's interactions. Hyperparameter tuning showed that moderate LSTM sizes (96–192 units) and dense layers (16–48 units) with low dropout (0–0.1) worked best. Very large layers or high dropout made learning harder, while moderate learning rates (~0.001–0.003) helped the model converge smoothly. This shows that the LSTM can capture the order of interactions well, even with a small dataset.

Unit VII

7. Limitations and Difficulties

A key **limitation** is the small and very specific sample size. The dataset includes only 24 children observed in similar play settings within the U.S., which means the findings cannot be taken as fully representative of the wider Deaf community worldwide. The results therefore provide useful insights into this particular group but should not be generalised too broadly**.** Some features, like whether the parent was Deaf or not, were very unbalanced in the dataset, which may have biased the results. Another limitation is that the models only predicted the average ASL activity for each child's session. This means they showed the overall level of signing but not the step-by-step changes that happened during the play. For example, the models could not tell if a child signed more at the start or increased later in the session. Because of this, the results give a broad picture but do not fully capture the detailed process of language development.

One of the main **difficulties** I faced was cleaning and preparing the data, since some files were inconsistent and needed manual fixing as some of the csv files did not have the index column. Another challenge was splitting the data for training and testing: for the LSTM I had to keep the sequences grouped by child, which made the process more complex. Balancing model complexity with the small dataset was also hard, since adding too many layers or parameters quickly led to overfitting. Finally, interpreting the metrics was sometimes challenging, as R^2 did not always reflect the low error values well due to the small range of ASL scores.

Unit VIII

8. Conclusion

8.1. Summary of Findings

My exploratory study assessed machine learning models in predicting sign language acquisition in children using cognitive and environmental features. The optimised Random Forest model performed best for aggregate-level predictions ($R^2 = 0.53$, RMSE = 43.77), highlighting key predictors such as age and interactional behaviours. This aligns with developmental research showing that both cognitive skills and environmental exposure influence early language learning (Tomasello, 1995; Meier, 1991).

The LSTM trained on sequence-level data, showed very low errors ($R^2 = 0.49$, RMSE = 0.02), demonstrating that temporal patterns in child–adult interactions can be informative. However, its performance was limited by the small sample size suggesting that with more data, LSTMs could capture richer dynamics of sign language development.

Linear Regression confirms that linear models struggle with complex patterns in ASL acquisition especially when it varies across children.

Overall, the study indicates that Random Forest provides reliable benchmarks for aggregate predictions while LSTM holds promise for capturing how children's ASL skills develop over time, in line with studies showing the potential of sequence-based models for language learning (Dupoux, 2016). Hence, my study supports the use of machine and deep learning models to further explore of language acquisition.

8.2. Applications and Future Works

The results of this study could help design predictive tools or software for early intervention programs allowing caregivers and educators to track children's sign language progress, provide feedback and suggest activities to support skill development. Sequence-based models, like LSTMs, could provide insights into how learning unfolds over time and suggests personalised strategies to improve sign language acquisition.

For future research, collecting more high-quality data from different places would improve deep learning performance and allow detailed per-interaction analysis in more general setting while collaborating with research group or health organisation. Adding other information such as facial expressions or environmental context and using advanced sequence models (e.g., Transformers, GRUs with attention) along with explainable AI methods like SHAP [Appendix R] could make these tools more accurate and easier to interpret for practical use in education and clinical settings.

Unit IX

9. Appendices

Appendix A: Traditionally, sign language acquisition has been studied through observational methods. While these studies have given valuable insights, they often lack the scale or precision needed to uncover deeper trends (Anderson & Reilly, 2002). With the growing availability of behavioural and interaction data, new opportunities are emerging to use machine learning and statistical modelling to get the idea how children acquire sign language and utilise it for their improvement.

Appendix B: TABLE-I shows a comparison of literature review to understand how my study is connected with other work. In the study by Greff et al. (2017), LSTM models were tested on different tasks that involve learning from sequences, like speech and handwriting. Instead of showing results like MSE or R^2, they focused more on comparing different versions of LSTM and how well they worked using task-specific loss metrics, such as cross-entropy loss for speech data. On the other hand, Lipnevich et al. (2016) used regression to look at how student behaviours affect learning outcomes. They focused more on which features were important rather than how accurate the predictions were.

Appendix C: The ASL-PLAY dataset was collected across multiple U.S. sites, where each 15-minute parent–child play session was coded for approximately 12 minutes by trained Deaf ASL researchers using ELAN (Crasborn & Sloetjes, 2008). The coding captured behaviours such as eye gaze, object touch, ASL signs and attention-getters and was exported as .csv files where each row represented a behaviour in time. These session files were merged with demographic data (e.g., child's age, parental hearing status, spoken English use) to provide richer context for analysis. All data were collected with informed consent under ethical approval and are publicly available. While a second release (Play2.0) exists with updated demographics, it focused only on novel object labelling, which limited comparability with Play1.0 and reduced model performance, so only Session 1 data were used in this project.

Appendix D: FIG-1 flowchart summarizes the preprocessing pipeline, where raw interaction files were merged, cleaned and engineered into the ASL_Score target variable before scaling and modelling.

Appendix E: ASL_Activity was created to capture moment-to-moment engagement of children during play (e.g., gaze, touch, vocalisation). Since ASL learning relies on interactional dynamics, this feature reflects active participation at each time step.

ASL_Score was designed as a summary outcome measure at the child level, combining multiple interactions across sessions to represent each child's overall sign learning progress. These features align with child language research, where both micro-level behaviours and macro-level outcomes are critical to understanding acquisition patterns (Tomasello, 2003; Kuhl, 2004).

Appendix F: FIG-2 frequency distribution of ASL_Score reveals a right-skewed pattern with most values clustered at the lower end which ranges between 0 -50.

Appendix G: FIG-3 box plot of ASL_Score highlights a narrow interquartile range and a low median, suggesting limited score variability with presence of outliers.

Appendix H: FIG-4 scatter plot showing a positive correlation between Child_Age and ASL_Score, with older participants tending to score higher.

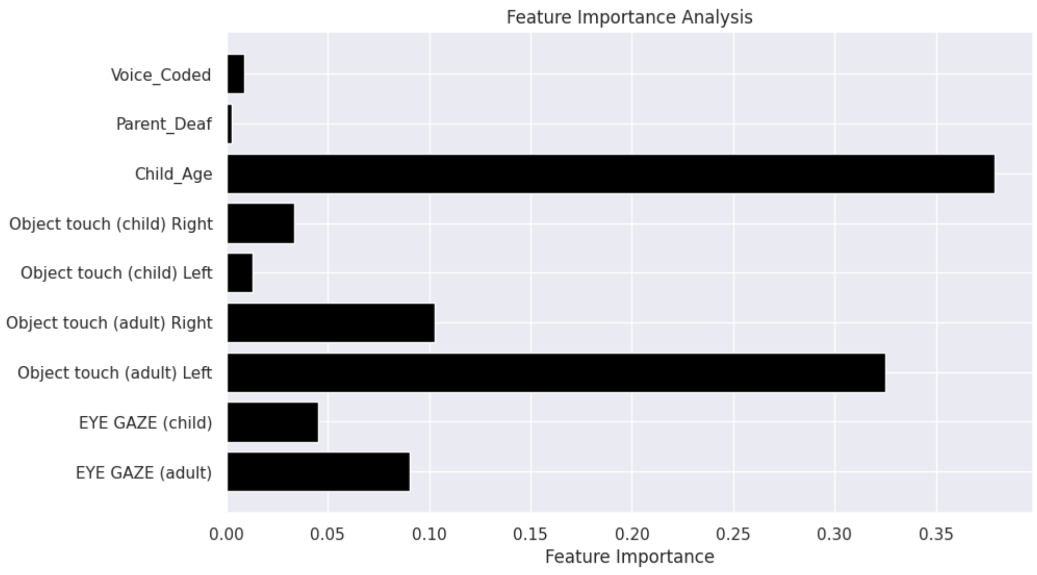
Appendix I: FIG-5 bar chart showing significantly higher ASL_Score among children with deaf parents, highlighting the impact of parental deaf status on language proficiency.

Appendix J: ReLU is an activation function defined by, f(x)=max(0,x) that adds non-linearity and helps the model learn complex patterns efficiently.

Appendix K: Table II compares the performance metrics of three machine learning models—Linear Regression, Random Forest and LSTM using RMSE, MAE, MSE and R^2. Linear Regression shows the highest error and a strongly negative R^2, symbolise poor performance. Random Forest performs moderately well with lower errors and a positive R^2. LSTM has extremely low error values but a near-zero negative R^2, suggesting possible overfitting or lack of generalization.

Appendix L: In the code snippet, I ranked all predictors using the Random Forest feature importances and kept only those above 0.05; this cut-off keeps the strongest signals (e.g., *Child_Age*, adult object-touch) while dropping weak, noisy inputs, which helps to avoid having too many overlapping predictors that carry the same information. Next, I checked the target ASL_Score for outliers with the Interquartile Range (IQR) rule: I computed Q1 and Q3, set bounds at *Q1 − 1.5×IQR* and *Q3 + 1.5×IQR*, and removed rows where the target fell outside these bounds. This prevents a few extreme scores from pulling the regression line and inflating error. Finally, I recreated a filtered dataset containing only the selected predictors and the non-outlier target values; this cleaner, lower-noise table is what I used for the optimisation of linear regression.

Appendix M: The feature Importance showed that Child_Age was the most important predictor because age reflects overall development. As children grow, they gain stronger attention, memory and motor skills, which naturally support learning ASL (Tomasello, 2003). Adult object touch was also highly important, since when adults handle or point to objects, it gives children a clear link between the sign and its meaning, helping them learn faster. Interestingly, object touches with the left hand showed more importance than the right hand, which likely reflects how adults interacted in the dataset (e.g., holding or supporting with the right hand and using the left hand for play) rather than a true learning difference. Gaze and voice features were less useful for prediction. This is likely because almost all children and parents look at each other during play. The Parent_Deaf variable showed low feature importance, likely because it is a binary factor with limited variance and weaker direct association with ASL scores compared to continuous features such as child age or interaction measures.



Appendix N: The code initializes a dictionary named param_dist to specify possible value ranges for important hyperparameters. RandomizedSearchCV was used to explore variations in n_estimators, max_depth, max_features, min_samples_split, and min_samples_leaf across 5-fold cross-validation (Bergstra & Bengio, 2012).

The n_estimators range (100–500) allows exploration of different forest sizes, where more trees can improve accuracy but also increase training time. For max_features, options like 'auto', 'sqrt', and 'log2' help control how many features are considered at each split this affects both model diversity and speed, with 'sqrt' often performing well in practice. The max_depth values (10–50 and None) test both shallow and deep trees, helping to avoid overfitting while still capturing complex patterns. min_samples_split and min_samples_leaf are tuned to avoid overly specific splits and tiny leaf nodes, which can harm generalization; values like 2, 5, and 10 (for splits) and 1, 2, and 4 (for leaves) are commonly used to strike this balance. Finally, the bootstrap parameter is tested with both True and False to compare standard bagging with full sampling, which can influence variance and bias.

Constraints on tree depth and leaf size were deliberately introduced to prevent overfitting, consistent with Breiman's (2001) principle that Random Forests generalise well when variance is controlled. Unlike Linear Regression, Random Forests are robust to noisy and irrelevant features. Overall, this optimisation aimed to balance efficiency and generalisation, enabling the model to capture complex non-linear interactions in ASL acquisition data.

Appendix O: The LSTM model was improved using a mix of architecture changes, hyperparameter tuning, regularisation, and data handling. A Bidirectional LSTM was chosen so the model could learn patterns in both directions of the child–parent interaction. The number of LSTM units and dense layer units was adjusted to avoid making the model too simple (missing patterns) or too complex (overfitting to noise).

For hyperparameter tuning, I used a random search to test different combinations of settings such as units, dropout rates, dense layer size, and learning rate. This approach was more systematic and efficient than guessing values manually.

To prevent overfitting, dropout and recurrent dropout were added so the model did not rely too much on any single neuron. EarlyStopping is there to stop training when the validation loss doesn't improve anymore. The learning rate was also tuned carefully to help the model learn at a steady pace without crashing.

Finally, since children's interaction sequences had different lengths, padding was applied so all sequences had the same size, and the features were scaled so variables like age would not dominate over smaller features such as gaze or object touch.

Appendix P: FIG-6 bar chart comparing R^2 Scores before and after optimization across three models, with LSTM showing the most improvement from -0.00 to 0.49.

Appendix Q: Table III summarizing five hyperparameter tuning trials for LSTM models, showing variations in architecture and learning rate, with Trial 1 achieving the lowest validation error. Notably, models with higher dropout or excessive complexity showed poorer generalization, indicating that simpler architectures with moderate capacity and lower regularization performed better on this dataset.

Appendix R: SHAP (SHapley Additive exPlanations) is an explainable AI method that helps understand the output of machine learning models. It assigns each feature (e.g., Child Age, Eye Gaze) a "contribution value" showing how much it influenced a particular prediction. This makes complex models more interpretable and transparent (Lundberg & Lee, 2017).

Appendix S: The Actual vs Predicted scatter plots show that Linear Regression underfits the data, with large errors and weak alignment with the true values. Random Forest improves the fit by modelling non-linear patterns, while the LSTM provides the best alignment, reflecting its ability to capture sequential and contextual dependencies in the dataset.

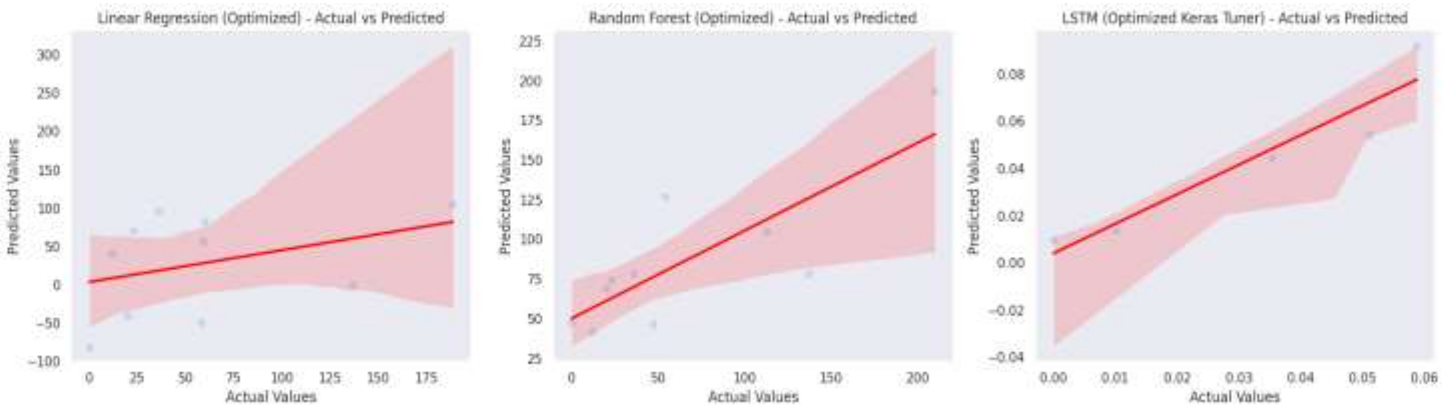


FIG 7: Correlation Scatter Plot of Three Models

Appendix T: The Actual vs Predicted line plots provide a qualitative view of model behaviour across individual samples.

Linear Regression shows large deviations from the true values, particularly at the extremes, suggesting it cannot capture the complexity of the data.

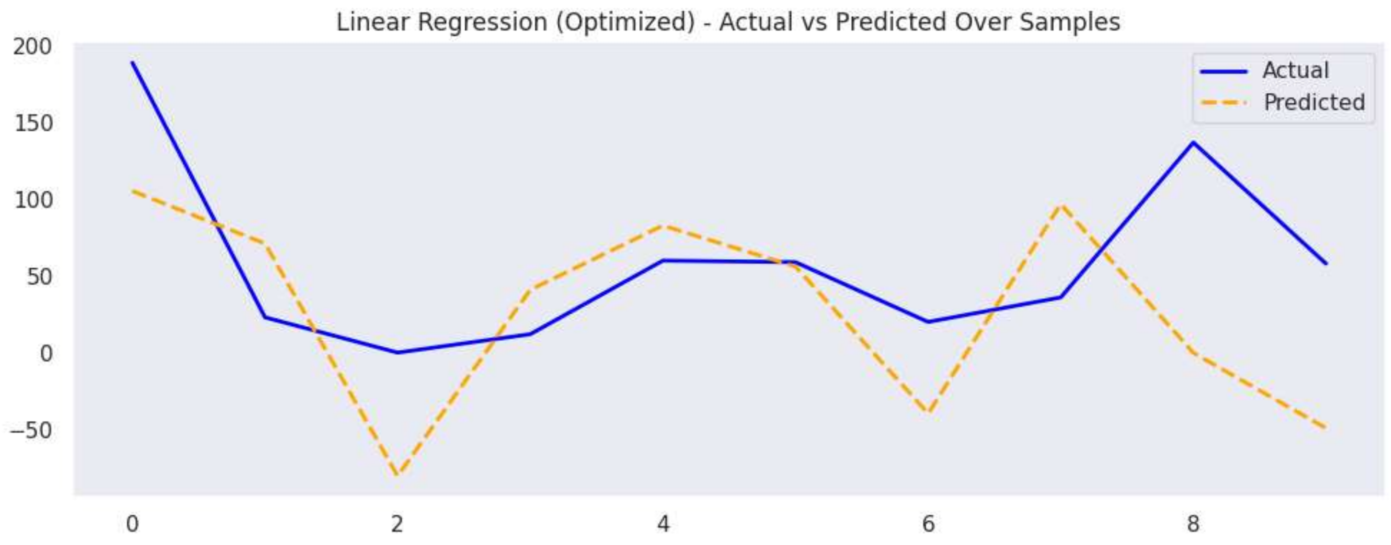


FIG 8: Actual Vs Predicted Linear Regression

Random Forest follows the overall trend more closely but still smooths out some sharp changes, indicating it is better suited for non-linear patterns but may overgeneralize in smaller datasets

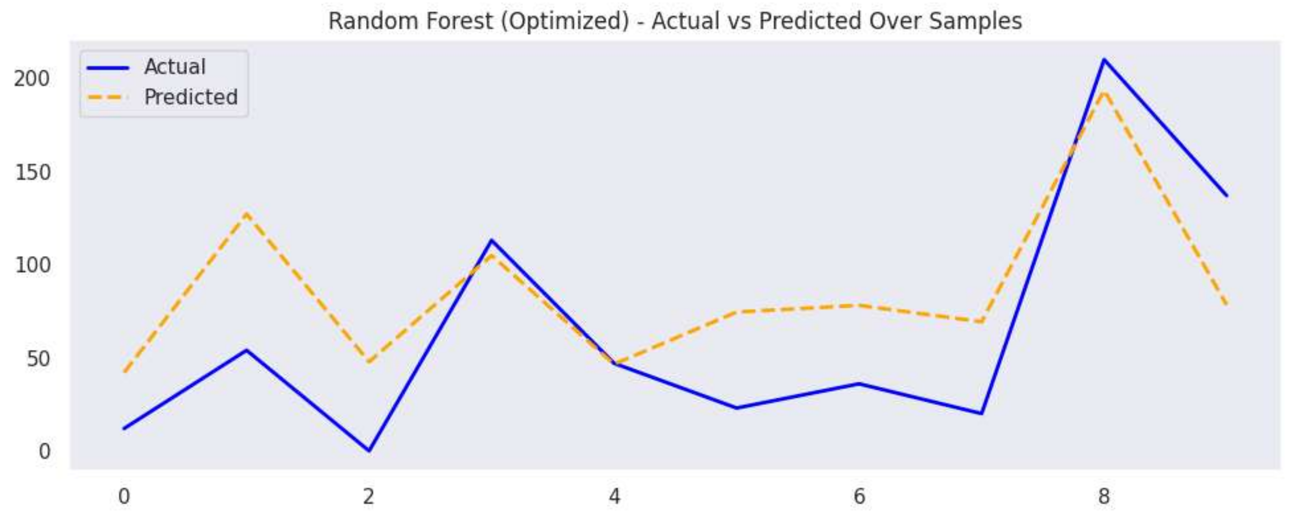


FIG 9: Actual Vs Predicted Random Forest

The LSTM shows the strongest alignment, with predicted and actual curves nearly overlapping. This reflects its advantage in modelling sequential structure, where small temporal or contextual cues accumulate into better predictive accuracy.

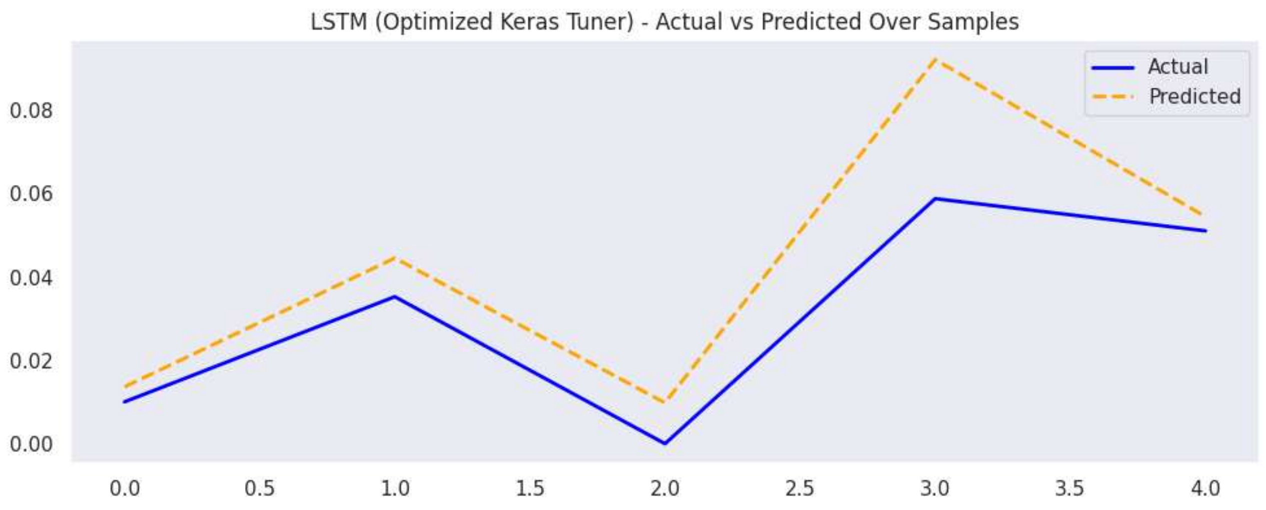


FIG 10: Actual Vs Predicted LSTM

Taken together, these plots highlight that while Random Forest improves over linear methods, deep learning approaches like LSTM can capture the nuanced, step-by-step process of ASL acquisition more effectively.

Appendix U:

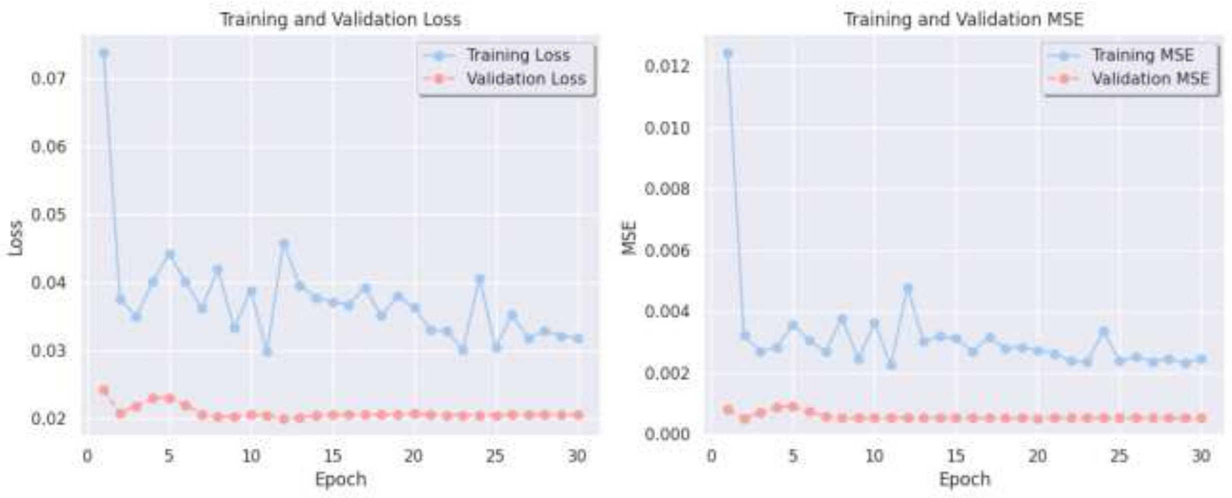


FIG 11: LSTM Training and Validation Curve

The plots above show the training and validation loss (MAE) on the left and the training and validation MSE on the right across 30 epochs.

- **Behaviour of Curves**: The training curves fluctuate more than the validation curves. One of the reasons can be small dataset size, where the model fits closely to the training data in some batches but less in others. The validation loss and MSE remain relatively low and stable, suggesting the model is not strongly overfitting at this stage but is limited in how much it can improve further.
- **Choice of Loss Function (MAE)**: Mean Absolute Error (MAE) was implemented as the loss function because it gives a clear measure of average prediction error. Compared to MSE, it's less affected by sudden outliers, making it helpful when children's ASL activity varies a lot.
- **Choice of Metric (MSE)**: Mean Squared Error (MSE) was chosen as the evaluation metric to complement MAE. Since MSE penalises larger errors more strongly, it helps highlight if the model occasionally makes large mistakes that MAE might mask. Using both together provides a balanced view of model performance.

In summary, these plots show that the model was able to learn sequential patterns without major overfitting, though the training instability reflects the small and variable nature of the dataset

Appendix V:

Here I have pasted my whole code from Jupyter Notebook which I have worked on for the fulfilment of my project.

Student Name - Tiasa Malitya

Student ID - 24005709

Project Title - Predictive Modelling of Sign Language Acquisition in Children

I have used three models Linear regression, Random Forest and LSTM to find out which model best works out to predict the sign language acquisition using ASL-PLAY dataset.

1. Import necessary libraries

The cell below contains the code to import all the essential ilbraries I need to access for successful completion of my project.

```
!pip install keras-tuner
```

```
Collecting keras-tuner
  Downloading keras_tuner-1.4.7-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: keras in /usr/local/lib/python3.11/dist-packages (from keras-tuner) (3.10.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from keras-tuner) (25.0)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from keras-tuner) (2.32.3)
Collecting kt-legacy (from keras-tuner)
  Downloading kt_legacy-1.0.5-py3-none-any.whl.metadata (221 bytes)
Requirement already satisfied: absl-py in /usr/local/lib/python3.11/dist-packages (from keras->keras-tuner) (1.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from keras->keras-tuner) (2.0.2)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras->keras-tuner) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras->keras-tuner) (0.1.0)
Requirement already satisfied: h5py in /usr/local/lib/python3.11/dist-packages (from keras->keras-tuner) (3.14.0)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras->keras-tuner) (0.17.0)
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.11/dist-packages (from keras->keras-tuner) (0.5.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->keras-tuner) (3.4.3)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->keras-tuner) (3.10)

Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->keras-tuner) (2.5.0)

Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->keras-tuner) (2025.8.3)

Requirement already satisfied: typing-extensions>=4.6.0 in /usr/local/lib/python3.11/dist-packages (from optree->keras->keras-tuner) (4.14.1)

Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras->keras-tuner) (4.0.0)

Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras->keras-tuner) (2.19.2)

Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras->keras-tuner) (0.1.2)

Downloading keras_tuner-1.4.7-py3-none-any.whl (129 kB)

   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
━ 129.1/129.1 kB 2.9 MB/s eta 0:00:00

Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)

Installing collected packages: kt-legacy, keras-tuner

Successfully installed keras-tuner-1.4.7 kt-legacy-1.0.5
```

```
from google.colab import drive
import os
import pandas as pd
import numpy as np
import glob
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout
from tensorflow.keras.callbacks import LearningRateScheduler, EarlyStopping
from sklearn.impute import SimpleImputer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import RandomizedSearchCV
import keras_tuner
from kerastuner.tuners import RandomSearch
from tensorflow.keras import layers, models
```

```
/tmp/ipython-input-3293833073.py:21: DeprecationWarning: `import kerastuner` is deprecated, please use `import keras_tuner`.
  from kerastuner.tuners import RandomSearch
```

```
# Mount Google Drive

drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Path to access dataset

session1 = "/content/drive/MyDrive/osfstorage-archive/1.0 Sessions"
demographics_file = r"/content/drive/MyDrive/osfstorage-archive/Demographic_Info.csv"
```

1 ***2. Steps into Preprocessing***

```
# Join csv files
csv_files = glob.glob(os.path.join(session1, "*.csv"))
dataframes = []
for file in csv_files:
   df = pd.read_csv(file, encoding='utf-8-sig')                    # Read the CSV
   df.columns = df.columns.str.strip()                             # Clean column names
   raw_name = os.path.splitext(os.path.basename(file))[0]
   child_id = raw_name.split('_Exported')[0]
   if 'Child ID' not in df.columns or df['Child ID'].isnull().all():         # Inject 'Child ID' into all rows
if missing or all null
       df['Child ID'] = [child_id] * len(df)                       # Assign to all rows
       print(f"Injected Child ID into: {os.path.basename(file)}")
   dataframes.append(df)                                           # Append to list

# Combine all cleaned data
combined_df = pd.concat(dataframes, ignore_index=True, sort=False)
```

```
Injected Child ID into: ASLPLAY_21_Exported_05_31_24.csv
Injected Child ID into: ASLPLAY_18_Exported_05_31_24.csv
Injected Child ID into: ASLPLAY_22_Exported_05_31_24.csv
Injected Child ID into: ASLPLAY_19_Exported_05_31_24.csv
Injected Child ID into: ASLPLAY_24_Exported_05_31_24.csv
Injected Child ID into: ASLPLAY_23_Exported_05_31_24.csv
Injected Child ID into: ASLPLAY_17_Exported_05_31_24.csv
```

```
# Load demographics CSV
demographics = pd.read_csv(demographics_file)
```

```
demographics.columns = demographics.columns.str.strip()                    # Remove any space in header




# Merge combined_df with demographics

merged_data = pd.merge(combined_df, demographics, on='Child ID', how='left')

merged_data.to_csv(r"C:\Users\tiasa\Downloads\merged_data.csv", index=False)    # Save the merge file




# Dataset summary

print("Unique Child IDs:", sorted(merged_data['Child ID'].unique()))

print("Final merged dataset shape:", merged_data.shape)

print("Columns:", merged_data.columns)

print("First five rows:", merged_data.head())
```

```
Unique Child IDs: ['ASLPLAY_01', 'ASLPLAY_02', 'ASLPLAY_03', 'ASLPLAY_04',
'ASLPLAY_05', 'ASLPLAY_06', 'ASLPLAY_07', 'ASLPLAY_08', 'ASLPLAY_09',
'ASLPLAY_10', 'ASLPLAY_11', 'ASLPLAY_12', 'ASLPLAY_13', 'ASLPLAY_14',
'ASLPLAY_15', 'ASLPLAY_16', 'ASLPLAY_17', 'ASLPLAY_18', 'ASLPLAY_19',
'ASLPLAY_20', 'ASLPLAY_21', 'ASLPLAY_22', 'ASLPLAY_23', 'ASLPLAY_24']

Final merged dataset shape: (40265, 74)

Columns: Index(['Child ID', 'Begin Time - ss.msec', 'End Time - ss.msec',

       'Duration - ss.msec', 'Adult Attention Getting Strategies',

       'Adult1 ASL Dominant hand', 'Adult1 Dominant Hand append', 'Adult1 NMS',

       'Adult1 Non-Dominant Hand append', 'Adult1 Non-Dominant hand',

       'Adult1 free translation', 'Child ASL Dominant hand',
       'Child ASL Dominant hand append', 'Child ASL Non-Dominant hand',
       'Child Attention Getting Strategies', 'Child NMS',
       'Child Non-Dominant Hand append', 'Child Spoken English Transcription',
       'Child Spoken English append', 'Child free translation',
       'EYE GAZE (adult)', 'EYE GAZE (adult) Append', 'EYE GAZE (child)',
       'EYE GAZE (child) Append', 'Object touch (adult) Left',
       'Object touch (adult) Left Append', 'Object touch (adult) Right',
       'Object touch (adult) Right Append', 'Object touch (child) Left',
       'Object touch (child) Left Append', 'Object touch (child) Right',
       'Object touch (child) Right Append',
       'Parent Spoken English Transcription', 'Parent Spoken English append',
       'Adult Attention Getting Strategies Append',
       'Child Attention Getting Strategies Append', 'Adult1 English',
       'Attention Getting Strategies', 'Child ASL feedback',
       'Adult1 ASL feedback', 'Child comments', 'Adult1 comments',
       'Checking comments', 'Child ASL Dominant hand pho',
       'Child ASL Non-Dominant pho', 'Adult1 ASL Dominant Hand pho',
       'Adult1 ASL Non-Dominant Hand pho', 'CSET Noun', 'PSET Referent',
       'CSET Noun2', 'PSET Ref2', 'CSET Referent', 'CSET Ref2',
       'Adult2 ASL Dominant hand', 'Adult2 ASL feedback',
       'Adult2 Dominant Hand append', 'Adult2 NMS',
       'Adult2 Non-Dominant Hand append', 'Adult2 Non-Dominant hand',
       'Adult2 free translation', 'EYE GAZE (adult2)',
       'EYE GAZE (adult2) Append', 'Object touch (adult2) Left',
       'Object touch (adult2) Right', 'Comments', 'Eric's Comments',
       'Adult2 comments', 'Adult2 Attention Getting Strategies',
       'Object touch (adult2) Left Append',
       'Object touch (adult2) Right Append', 'Child Spoken English',
       'Child_Age', 'Parent_Deaf', 'Voice_Coded'],
       dtype='object')
```

```
First five rows:     Child ID  Begin Time - ss.msec  End Time - ss.msec  Duration - ss.msec  \
0  ASLPLAY_03              64.341              64.793               0.452
1  ASLPLAY_03              64.793              65.094               0.301
2  ASLPLAY_03              65.790              67.179               1.389
3  ASLPLAY_03              66.733              67.179               0.446
4  ASLPLAY_03              68.704              69.361               0.657

  Adult Attention Getting Strategies Adult1 ASL Dominant hand  \
0                                NaN                      NaN
1                                NaN                      NaN
2                                NaN                      NaN
3                                NaN                  BYE-BYE
4                                NaN                      NaN

  Adult1 Dominant Hand append Adult1 NMS Adult1 Non-Dominant Hand append  \
0                         NaN        NaN                             NaN
1                         NaN        NaN                             NaN
2                         NaN        NaN                             NaN
3                         NaN        NaN                             NaN
4                         NaN        NaN                             NaN

  Adult1 Non-Dominant hand  ... Comments Eric's Comments Adult2 comments  \
0                      NaN  ...      NaN            NaN             NaN
1                      NaN  ...      NaN            NaN             NaN
2                      NaN  ...      NaN            NaN             NaN
3                      NaN  ...      NaN            NaN             NaN
4                      NaN  ...      NaN            NaN             NaN

  Adult2 Attention Getting Strategies Object touch (adult2) Left Append  \
0                                 NaN                               NaN
```

```
1                    NaN                  NaN
2                    NaN                  NaN
3                    NaN                  NaN
4                    NaN                  NaN

  Object touch (adult2) Right Append Child Spoken English Child_Age  \
0                    NaN              NaN        16
1                    NaN              NaN        16
2                    NaN              NaN        16
3                    NaN              NaN        16
4                    NaN              NaN        16

  Parent_Deaf Voice_Coded
0       0       1
1       0       1
2       0       1
3       0       1
4       0       1

[5 rows x 74 columns]
```

2 ***3. Feature Engineering***

I have created the target variable as ASL_Score

```
# Selected Features to get ASL_Score
asl_columns = [
    'Child ASL Dominant hand',
    'Child ASL Non-Dominant hand',
    'Child ASL Dominant hand pho',
    'Child ASL Non-Dominant pho',
    'Child NMS',
    'Child ASL feedback',
    'Child free translation'
              ]
merged_data['ASL_Activity'] = merged_data[asl_columns].notna().sum(axis=1)      # Count non-null values across these columns per row
asl_scores = merged_data.groupby('Child ID')['ASL_Activity'].sum().reset_index()# Sum by Child ID
asl_scores = asl_scores.rename(columns={'ASL_Activity': 'ASL_Score'})
merged_data = pd.merge(merged_data, asl_scores, on='Child ID', how='left')
print(asl_scores.sort_values(by='ASL_Score', ascending=False))                # Print ASL_Score
```

```
      Child ID  ASL_Score
20  ASLPLAY_21        365
22  ASLPLAY_23        251
21  ASLPLAY_22        210
15  ASLPLAY_16        189
23  ASLPLAY_24        168
5   ASLPLAY_06        137
18  ASLPLAY_19        113
14  ASLPLAY_15         97
19  ASLPLAY_20         78
17  ASLPLAY_18         60
12  ASLPLAY_13         59
2   ASLPLAY_03         58
16  ASLPLAY_17         54
11  ASLPLAY_12         47
7   ASLPLAY_08         44
6   ASLPLAY_07         37
13  ASLPLAY_14         36
```

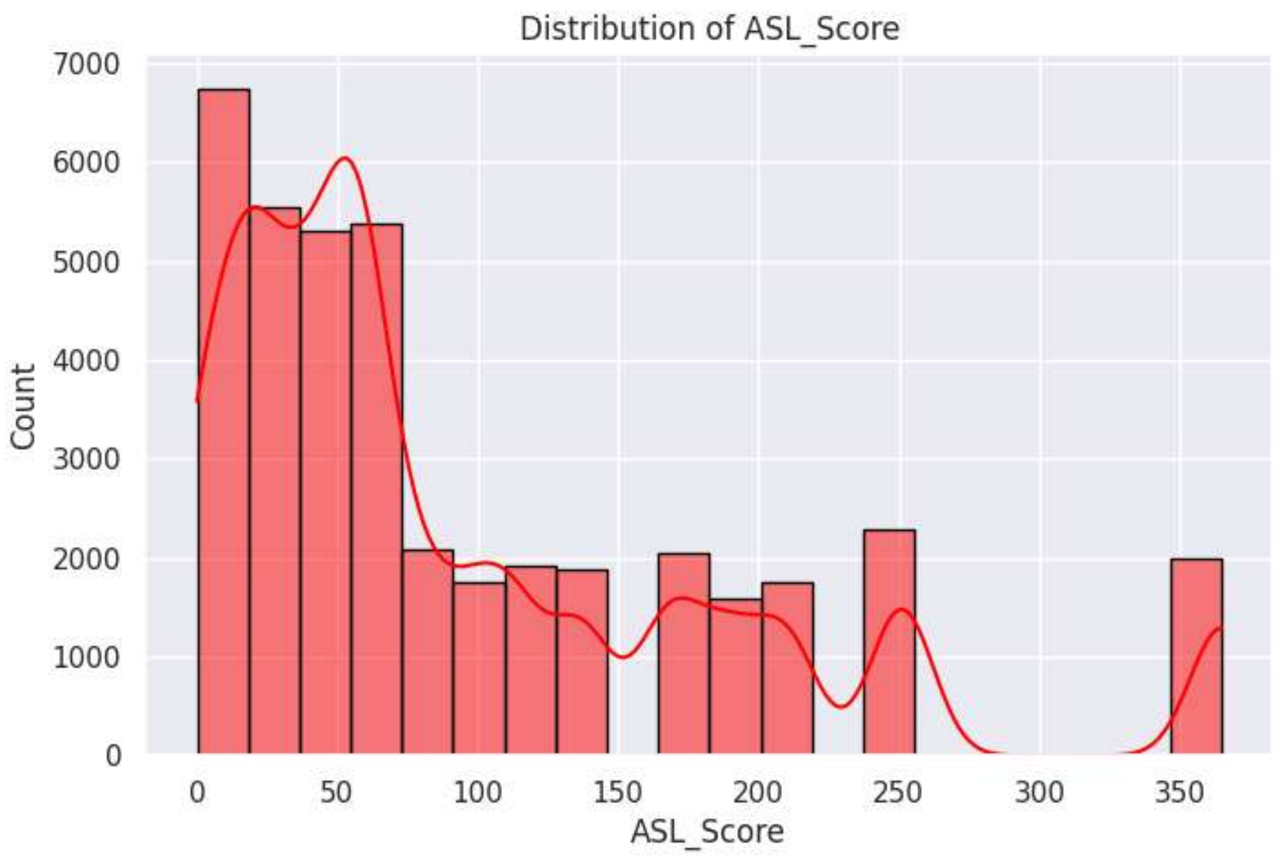
```
9   ASLPLAY_10       23
1   ASLPLAY_02       20
10  ASLPLAY_11        17
3   ASLPLAY_04       12
8   ASLPLAY_09       12
4   ASLPLAY_05        0
0   ASLPLAY_01        0
```

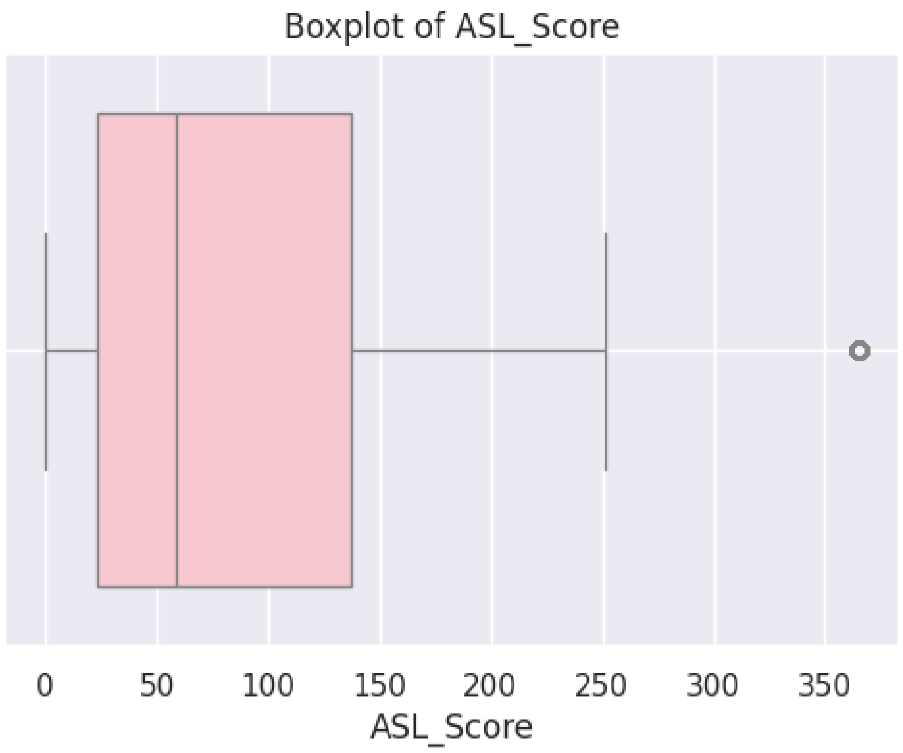
3 ***4. Exploratory Data Analysis (EDA)***

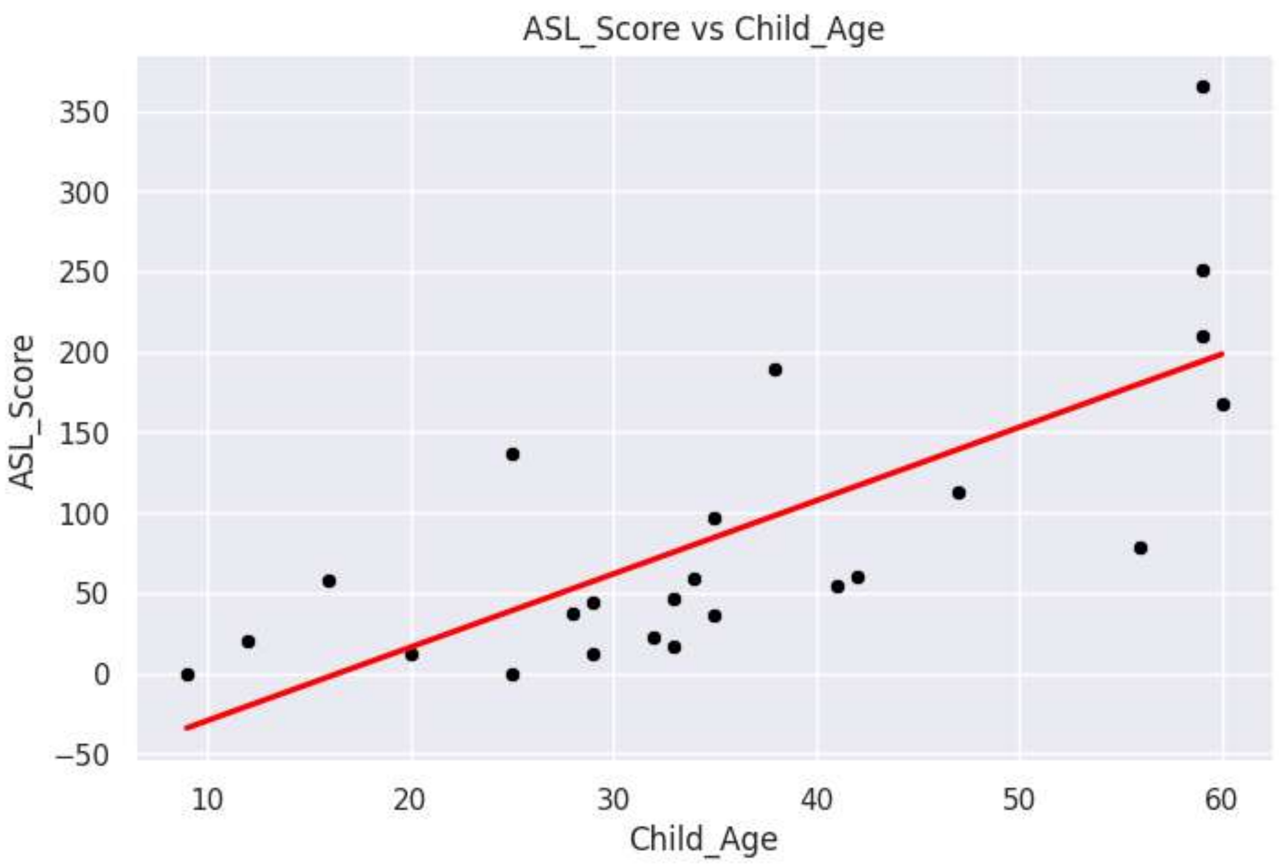
In this cell, I have visualized some important plots to explore the distribution of features and their relation with the target variable.

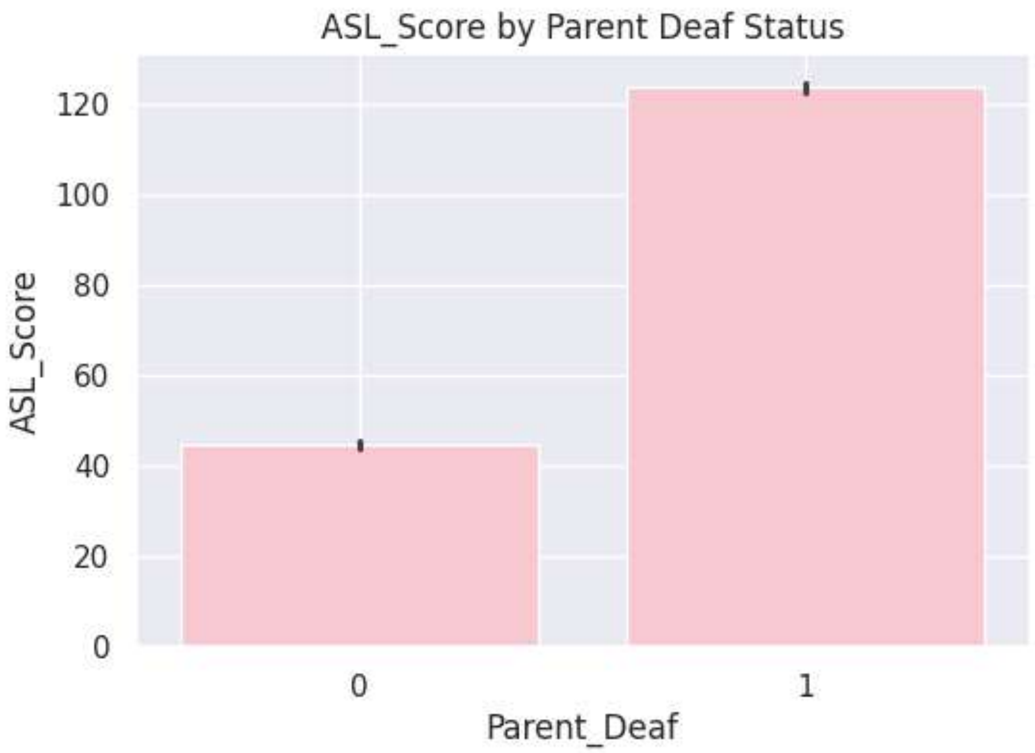
```
# Set up seaborn style
sns.set(style='dark', palette='pastel')

# Distribution of ASL_Score
plt.figure(figsize=(8, 5))
sns.histplot(merged_data['ASL_Score'], kde=True, bins=20, color= "red", edgecolor="black")
plt.title("Distribution of ASL_Score")
plt.xlabel("ASL_Score")
plt.ylabel("Count")
plt.grid("dashed")
plt.savefig("Distribution_asl_score.jpg", dpi=300)

# Boxplot to check for outliers
plt.figure(figsize=(6, 4))
sns.boxplot(x=merged_data['ASL_Score'], color="pink")
plt.title("Boxplot of ASL_Score")
plt.xlabel("ASL_Score")
plt.grid("dashed")
plt.savefig("Boxplot.jpg", dpi=300)
```

```
# ASL_Score vs Child Age
plt.figure(figsize=(8, 5))
sns.scatterplot(data=merged_data, x='Child_Age', y='ASL_Score', color="black")
sns.regplot(data=merged_data, x='Child_Age', y='ASL_Score', scatter=False, color='red')
plt.title("ASL_Score vs Child_Age")
plt.xlabel("Child_Age")
plt.ylabel("ASL_Score")
plt.grid("dashed")
plt.savefig("ASL_Score_vs_Child_Age.jpg", dpi=300)

# ASL_Score by Parent Deaf status
plt.figure(figsize=(6, 4))
sns.barplot(data=merged_data, x='Parent_Deaf', y='ASL_Score', color="pink")
plt.xlabel("Parent_Deaf")
plt.ylabel("ASL_Score")
plt.title("ASL_Score by Parent Deaf Status")
plt.grid("dashed")
plt.savefig("ASL_Score_by_Parent_Deaf.jpg", dpi=300)
```


Distribution of ASL_Score
Count
ASL_Score
0
1000
2000
3000
4000
5000
6000
7000
0
50
100
150
200
250
300
350


Boxplot of ASL_Score
ASL_Score
0
50
100
150
200
250
300
350





# Final Features selection to train the models

```
interaction_cols = [
   'EYE GAZE (adult)',
   'EYE GAZE (child)',
   'Object touch (adult) Left',
   'Object touch (adult) Right',
   'Object touch (child) Left',
   'Object touch (child) Right'
]
interaction_counts = merged_data.groupby('Child ID')[interaction_cols].apply(
                  lambda df: df.notnull().sum()).reset_index()              # count number of non-null entries per child per column
final_features = interaction_counts.merge(                                  # Merge with other features
              merged_data[['Child ID', 'Child_Age', 'Parent_Deaf',
              'Voice_Coded', 'ASL_Score']].drop_duplicates(),
              on='Child ID',
              how='left'
)
final_features
```

	Child ID	**EYE GAZE (adult)**	**EYE GAZE (child)**	**Object touch (adult) Left**	**Object touch (adult) Right**	**Object touch (child) Left**	**Object touch (child) Right**	**Child_Age**	**Parent_Deaf**	**Voice_Coded**	**ASL_Score**
0	ASLPLAY_01	407	319	109	138	69	67	9	0	1	0
1	ASLPLAY_02	425	298	82	92	68	75	12	0	1	20

	Child ID	EYE GAZE (adult)	EYE GAZE (child)	Object touch (adult) Left	Object touch (adult) Right	Object touch (child) Left	Object touch (child) Right	Child_Age	Parent_Deaf	Voice_Coded	ASL_Score
2	ASLPLAY_03	369	281	127	114	45	42	16	0	1	58
3	ASLPLAY_04	133	116	10	13	20	20	20	0	1	12
4	ASLPLAY_05	340	372	141	96	149	159	25	1	1	0
5	ASLPLAY_06	459	395	43	149	78	82	25	1	0	137
6	ASLPLAY_07	409	230	89	107	113	65	28	1	0	37
7	ASLPLAY_08	402	288	45	104	71	73	29	1	0	44
8	ASLPLAY_09	310	232	63	51	65	60	29	0	1	12
9	ASLPLAY_10	687	478	67	88	127	145	32	0	1	23
10	ASLPLAY_11	337	255	48	58	51	100	33	0	0	17
11	ASLPLAY_12	216	156	55	68	38	40	33	1	0	47
12	ASLPLAY_13	411	333	83	73	111	121	34	1	1	59

	Child ID	EYE GAZE (adult)	EYE GAZE (child)	Object touch (adult) Left	Object touch (adult) Right	Object touch (child) Left	Object touch (child) Right	Child_Age	Parent_Deaf	Voice_Coded	ASL_Score
13	ASLPLAY_14	693	298	44	71	70	63	35	0	1	36
14	ASLPLAY_15	450	352	95	80	87	105	35	1	0	97
15	ASLPLAY_16	440	316	30	33	70	74	38	1	0	189
16	ASLPLAY_17	453	375	42	64	97	84	41	1	1	54
17	ASLPLAY_18	441	496	79	94	168	120	42	1	0	60
18	ASLPLAY_19	396	386	57	50	60	84	47	1	0	113
19	ASLPLAY_20	370	337	73	71	132	103	56	1	1	78
20	ASLPLAY_21	420	446	5	6	78	114	59	1	0	365
21	ASLPLAY_22	418	359	20	43	87	124	59	1	0	210
22	ASLPLAY_23	342	327	20	34	104	81	59	1	0	251
23	ASLPLAY_24	501	485	36	83	109	89	60	0	0	168

4 *5. Regression Models*

```
# Define features and target
X = final_features.drop(columns=['Child ID', 'ASL_Score'])
y = final_features['ASL_Score']

# Split in train-test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, random_state=42)

# Scaling the variables
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train Linear Regression model
Lr_model = LinearRegression()
Lr_model.fit(X_train_scaled, y_train)
Lr_y_pred = Lr_model.predict(X_test_scaled)

# Evaluation
Lr_mae = mean_absolute_error(y_test, Lr_y_pred)
Lr_mse = mean_squared_error(y_test, Lr_y_pred)
Lr_rmse = np.sqrt(Lr_mse)
Lr_R2 = r2_score(y_test, Lr_y_pred)

print(f"Linear Regression MAE: {Lr_mae:.2f},MSE: {Lr_mse:.2f},RMSE: {Lr_rmse:.2f},R²:
{Lr_R2:.2f}")

# Train Random Forest model
```

```
Rf_model = RandomForestRegressor(random_state=42)
Rf_model.fit(X_train, y_train)
Rf_y_pred = Rf_model.predict(X_test)

# Evaluation
Rf_mae = mean_absolute_error(y_test, Rf_y_pred)
Rf_mse = mean_squared_error(y_test, Rf_y_pred)
Rf_rmse = np.sqrt(Rf_mse)
Rf_R2 = r2_score(y_test, Rf_y_pred)

print(f"Random Forest MAE: {Rf_mae:.2f},MSE: {Rf_mse:.2f},RMSE: {Rf_rmse:.2f}, R²:
{Rf_R2:.2f}")
```

```
Linear Regression MAE: 80.06,MSE: 9157.54,RMSE: 95.70,R²: -1.25
Random Forest MAE: 41.98,MSE: 2223.28,RMSE: 47.15, R²: 0.45
```

5 ***6. Long Short-Term Memory Model***

```
# Load dataset
data_path = '/content/drive/MyDrive/merged_data.csv'
merged_data = pd.read_csv(data_path)

# Select features
feature_cols = [
    'EYE GAZE (adult)', 'EYE GAZE (child)',
    'Object touch (adult) Left', 'Object touch (adult) Right',
    'Object touch (child) Left', 'Object touch (child) Right',
    'Child_Age', 'Parent_Deaf', 'Voice_Coded'
]

data = merged_data[['Child ID'] + feature_cols + ['ASL_Activity']].copy()
```

```
# Clean and impute
for col in feature_cols:
    data[col] = pd.to_numeric(data[col], errors='coerce')
imputer = SimpleImputer(strategy='constant', fill_value=0)
data[feature_cols] = imputer.fit_transform(data[feature_cols])

# Scale
scaler = StandardScaler()
data[feature_cols] = scaler.fit_transform(data[feature_cols])

# Group into sequences
sequence_data = []
sequence_targets = []

for _, group in data.groupby('Child ID'):
    X_seq = group[feature_cols].values
    y_seq = group['ASL_Activity'].values
    if len(X_seq) >= 10:
        sequence_data.append(X_seq)
        # Predict average ASL activity per child sequence
        sequence_targets.append(np.mean(y_seq))

# Pad X sequences (y is already 1D now)
X_padded = pad_sequences(sequence_data, padding='post', dtype='float32')
y_array = np.array(sequence_targets, dtype='float32')

# Train-test split
Lstm_X_train, Lstm_X_test, Lstm_y_train, Lstm_y_test = train_test_split(X_padded, y_array,
test_size=0.2, random_state=42)
```

```
# LSTM model
Lstm_model = Sequential([
    Input(shape=(Lstm_X_train.shape[1], Lstm_X_train.shape[2])),
    LSTM(64, return_sequences=False),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1)
])

Lstm_model.compile(optimizer='adam', loss='mae', metrics=['mse'])

# LR scheduler
def scheduler(epoch, lr):
    return lr if epoch < 10 else float(lr * tf.math.exp(-0.1))

lr_callback = LearningRateScheduler(scheduler)

# Train
history = Lstm_model.fit(
    Lstm_X_train, Lstm_y_train,
    validation_data=(Lstm_X_test, Lstm_y_test),
    epochs=30,
    batch_size=8,                                    # increase batch size to 8
    callbacks=[lr_callback],
    verbose=1
)
```

```
/tmp/ipython-input-1751185262.py:3: DtypeWarning: Columns
(7,14,17,26,36,37,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,60,61,62,63,64,65,6
7,68,69,70) have mixed types. Specify dtype option on import or set low_memory=False.
```

```
  merged_data = pd.read_csv(data_path)
/usr/local/lib/python3.11/dist-packages/sklearn/impute/_base.py:572: FutureWarning: Currently, when `keep_empty_feature=False` and `strategy="constant"`, empty features are not dropped. This behaviour will change in version 1.8. Set `keep_empty_feature=True` to preserve this behaviour.
  warnings.warn(
Epoch 1/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 496ms/step - loss: 0.0688 - mse: 0.0106 - val_loss: 0.0242 - val_mse: 8.0248e-04 - learning_rate: 0.0010
Epoch 2/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 410ms/step - loss: 0.0346 - mse: 0.0029 - val_loss: 0.0207 - val_mse: 5.1941e-04 - learning_rate: 0.0010
Epoch 3/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 406ms/step - loss: 0.0356 - mse: 0.0029 - val_loss: 0.0218 - val_mse: 7.0267e-04 - learning_rate: 0.0010
Epoch 4/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 406ms/step - loss: 0.0417 - mse: 0.0033 - val_loss: 0.0230 - val_mse: 8.8514e-04 - learning_rate: 0.0010
Epoch 5/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 410ms/step - loss: 0.0395 - mse: 0.0026 - val_loss: 0.0230 - val_mse: 8.9776e-04 - learning_rate: 0.0010
Epoch 6/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 540ms/step - loss: 0.0408 - mse: 0.0029 - val_loss: 0.0220 - val_mse: 7.2262e-04 - learning_rate: 0.0010
Epoch 7/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 377ms/step - loss: 0.0375 - mse: 0.0028 - val_loss: 0.0206 - val_mse: 5.7433e-04 - learning_rate: 0.0010
Epoch 8/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 411ms/step - loss: 0.0345 - mse: 0.0029 - val_loss: 0.0203 - val_mse: 5.2717e-04 - learning_rate: 0.0010
Epoch 9/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 415ms/step - loss: 0.0314 - mse: 0.0023 - val_loss: 0.0202 - val_mse: 5.2945e-04 - learning_rate: 0.0010
Epoch 10/30
```

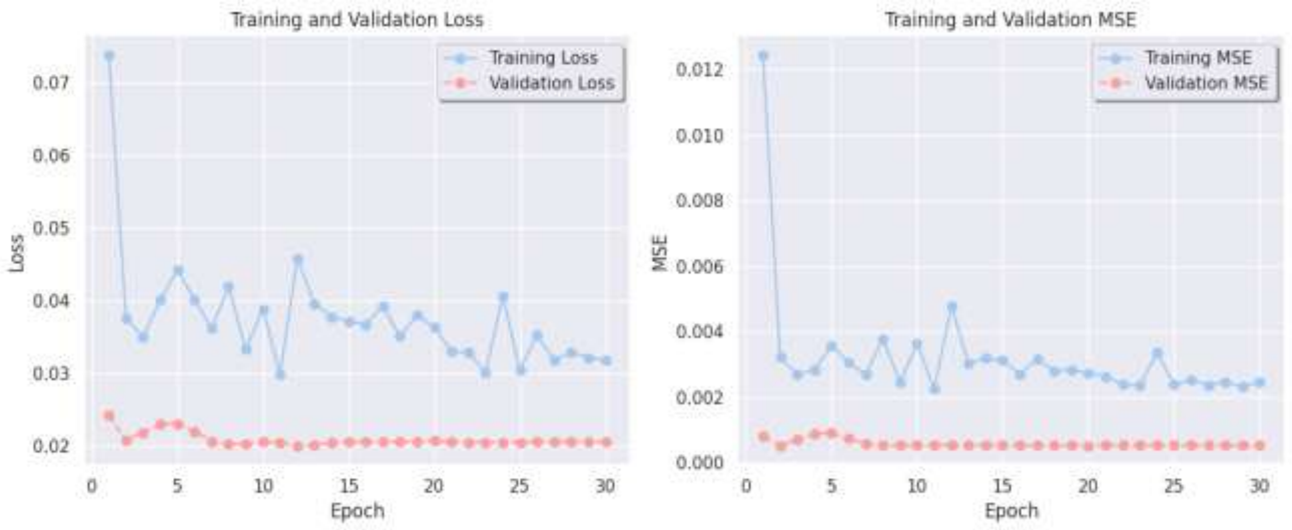
```
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 410ms/step - loss: 0.0397 - mse: 0.0040 -
val_loss: 0.0205 - val_mse: 5.2169e-04 - learning_rate: 0.0010

Epoch 11/30

3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 405ms/step - loss: 0.0258 - mse: 0.0016 -
val_loss: 0.0203 - val_mse: 5.2497e-04 - learning_rate: 9.0484e-04

Epoch 12/30

3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 403ms/step - loss: 0.0519 - mse: 0.0058 -
val_loss: 0.0200 - val_mse: 5.3893e-04 - learning_rate: 8.1873e-04

Epoch 13/30

3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 402ms/step - loss: 0.0378 - mse: 0.0028 -
val_loss: 0.0200 - val_mse: 5.3439e-04 - learning_rate: 7.4082e-04

Epoch 14/30

3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 424ms/step - loss: 0.0387 - mse: 0.0034 -
val_loss: 0.0204 - val_mse: 5.2426e-04 - learning_rate: 6.7032e-04

Epoch 15/30

3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 798ms/step - loss: 0.0364 - mse: 0.0028 -
val_loss: 0.0205 - val_mse: 5.2159e-04 - learning_rate: 6.0653e-04

Epoch 16/30

3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 406ms/step - loss: 0.0314 - mse: 0.0020 -
val_loss: 0.0205 - val_mse: 5.2220e-04 - learning_rate: 5.4881e-04

Epoch 17/30

3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 398ms/step - loss: 0.0416 - mse: 0.0036 -
val_loss: 0.0205 - val_mse: 5.2203e-04 - learning_rate: 4.9659e-04

Epoch 18/30

3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 368ms/step - loss: 0.0320 - mse: 0.0024 -
val_loss: 0.0205 - val_mse: 5.2125e-04 - learning_rate: 4.4933e-04

Epoch 19/30

3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 416ms/step - loss: 0.0351 - mse: 0.0024 -
val_loss: 0.0206 - val_mse: 5.2052e-04 - learning_rate: 4.0657e-04

Epoch 20/30

3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 376ms/step - loss: 0.0339 - mse: 0.0024 -
val_loss: 0.0206 - val_mse: 5.2006e-04 - learning_rate: 3.6788e-04

Epoch 21/30
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 411ms/step - loss: 0.0373 - mse: 0.0034 - val_loss: 0.0205 - val_mse: 5.2204e-04 - learning_rate: 3.3287e-04
Epoch 22/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 411ms/step - loss: 0.0345 - mse: 0.0027 - val_loss: 0.0205 - val_mse: 5.2264e-04 - learning_rate: 3.0119e-04
Epoch 23/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 408ms/step - loss: 0.0253 - mse: 0.0018 - val_loss: 0.0205 - val_mse: 5.2247e-04 - learning_rate: 2.7253e-04
Epoch 24/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 716ms/step - loss: 0.0402 - mse: 0.0032 - val_loss: 0.0204 - val_mse: 5.2278e-04 - learning_rate: 2.4660e-04
Epoch 25/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 416ms/step - loss: 0.0296 - mse: 0.0022 - val_loss: 0.0205 - val_mse: 5.2263e-04 - learning_rate: 2.2313e-04
Epoch 26/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 410ms/step - loss: 0.0378 - mse: 0.0029 - val_loss: 0.0205 - val_mse: 5.2170e-04 - learning_rate: 2.0190e-04
Epoch 27/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 409ms/step - loss: 0.0290 - mse: 0.0020 - val_loss: 0.0205 - val_mse: 5.2163e-04 - learning_rate: 1.8268e-04
Epoch 28/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 404ms/step - loss: 0.0388 - mse: 0.0032 - val_loss: 0.0206 - val_mse: 5.2076e-04 - learning_rate: 1.6530e-04
Epoch 29/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 400ms/step - loss: 0.0294 - mse: 0.0020 - val_loss: 0.0206 - val_mse: 5.2056e-04 - learning_rate: 1.4957e-04
Epoch 30/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 404ms/step - loss: 0.0341 - mse: 0.0025 - val_loss: 0.0205 - val_mse: 5.2157e-04 - learning_rate: 1.3534e-04
```

```
import matplotlib.pyplot as plt


def plot_training_history(history):
```

```
# Extract metrics
loss = history.history['loss']
val_loss = history.history['val_loss']
mse = history.history['mse']
val_mse = history.history['val_mse']
lr = history.history['learning_rate']
epochs = range(1, len(loss) + 1)

# Plot Loss
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, loss, 'bo-', label='Training Loss')
plt.plot(epochs, val_loss, 'ro--', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.xticks(ticks=range(0, len(loss) + 1, 5))
plt.grid("dashed")
plt.legend(shadow=True)

# Plot MAE
plt.subplot(1, 2, 2)
plt.plot(epochs, mse, 'bo-', label='Training MSE')
plt.plot(epochs, val_mse, 'ro--', label='Validation MSE')
plt.title('Training and Validation MSE')
plt.xlabel('Epoch')
plt.ylabel('MSE')
plt.xticks(ticks=range(0, len(loss) + 1, 5))
plt.grid("dashed")
plt.legend(shadow=True)
```

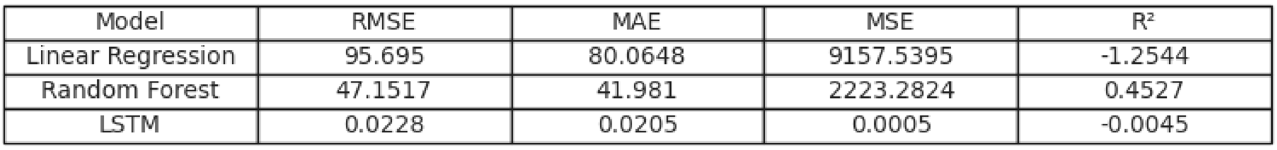
```
    plt.tight_layout()
    plt.savefig("Loss-Metric.jpg", dpi=300)

# Usage
plot_training_history(history)
```



```
# Predict
Lstm_y_pred = Lstm_model.predict(Lstm_X_test).flatten()

# Test Metrics
Lstm_mae = mean_absolute_error(Lstm_y_test, Lstm_y_pred)
Lstm_mse = mean_squared_error(Lstm_y_test, Lstm_y_pred)
Lstm_rmse = np.sqrt(Lstm_mse)
Lstm_R2 = r2_score(Lstm_y_test, Lstm_y_pred)

print("Test Metrics:")
print(f"  MAE:  {Lstm_mae:.4f}")
print(f"  MSE:  {Lstm_mse:.4f}")
print(f"  RMSE: {Lstm_rmse:.4f}")
print(f"  R²:   {Lstm_R2:.4f}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 182ms/step
Test Metrics:
  MAE:  0.0205
  MSE:  0.0005
  RMSE: 0.0228
  R²:   -0.0045
```

```
# Build comparision table
metrics_data = {
    'Model': ['Linear Regression', 'Random Forest', 'LSTM'],
    'RMSE': [Lr_rmse, Rf_rmse, Lstm_rmse],
    'MAE': [Lr_mae, Rf_mae, Lstm_mae],
    'MSE': [Lr_mse, Rf_mse, Lstm_mse],
    'R²': [Lr_R2, Rf_R2, Lstm_R2]
            }
df_metrics = pd.DataFrame(metrics_data)

# Display
fig, ax = plt.subplots(figsize=(8, 2))
ax.axis('tight')
ax.axis('off')
table = ax.table(
    cellText=df_metrics.round(4).values,
    colLabels=df_metrics.columns,
    cellLoc='center',
    loc='center'
)

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

plt.show()
plt.savefig("Comparison.jpg", dpi=300)

# Also print DataFrame for reference
print(df_metrics.round(4))
```



Model	RMSE	MAE	MSE	R²
Linear Regression	95.695	80.0648	9157.5395	-1.2544
Random Forest	47.1517	41.981	2223.2824	0.4527
LSTM	0.0228	0.0205	0.0005	-0.0045

```
             Model     RMSE      MAE        MSE       R²
0  Linear Regression  95.6950  80.0648  9157.5395 -1.2544
1      Random Forest  47.1517  41.9810  2223.2824  0.4527
2               LSTM   0.0228   0.0205     0.0005 -0.0045
<Figure size 640x480 with 0 Axes>
```

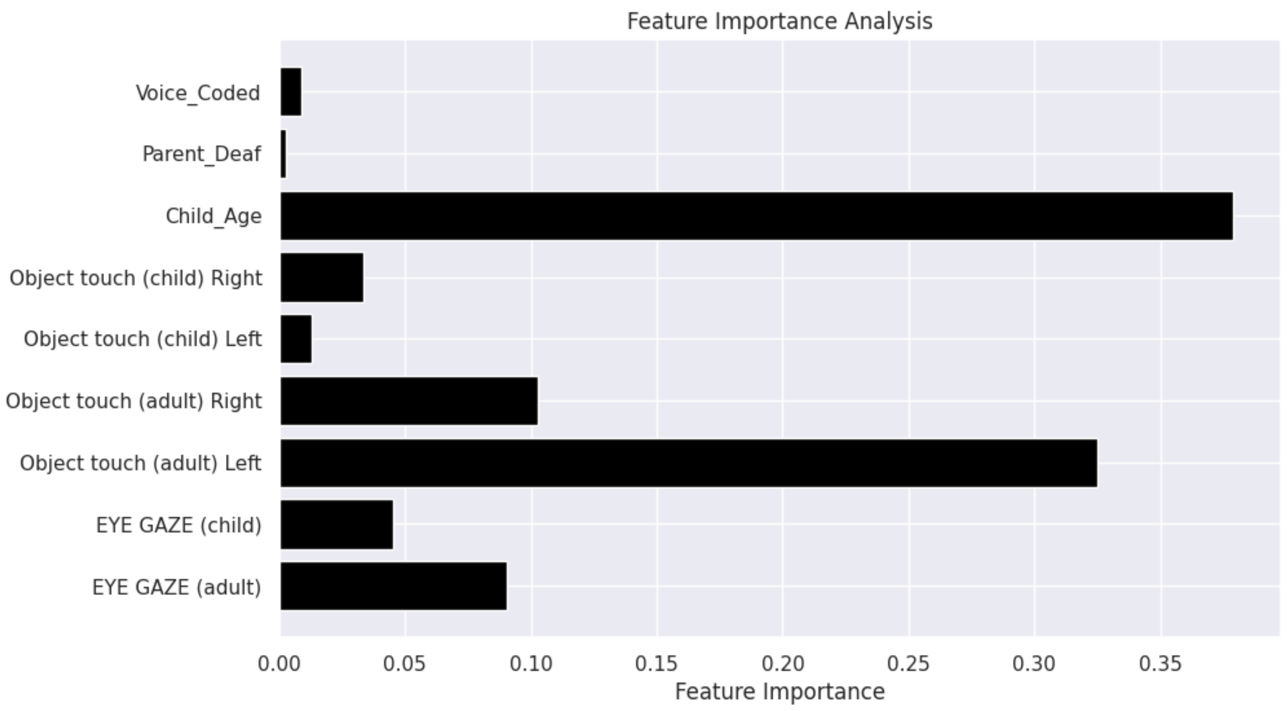
6 *7. Optimisation of the Models*

After building and evaluating the initial models, Now I have focused on optimizing their performance.

7 **Handling Outliers and Re-applying Linear Regression**

```
# Get feature importance from Random Forest
importances = Rf_model.feature_importances_
features = X.columns

plt.figure(figsize=(10, 6))                                    # Plot
plt.barh(features, importances, color="black")
plt.xlabel("Feature Importance")
plt.title("Feature Importance Analysis")
plt.grid("dashed")
plt.show()
plt.savefig("Feature-imp.jpg", dpi=300)

feature_importance_df = (                                      # Create DataFrame
   pd.DataFrame({
      'Feature': features,
      'Importance': importances
   })
   .sort_values(by='Importance', ascending=False)
)

print("\nRandom Forest Feature Importances")
print(feature_importance_df)

selected_features = feature_importance_df.loc[                         # Select features above
threshold
   feature_importance_df['Importance'] > 0.05, 'Feature'].tolist()
print(f"\nSelected Features: {selected_features}")

Q1, Q3 = final_features['ASL_Score'].quantile([0.25, 0.75])            # Outlier Removal for
Target (ASL_Score)
```

```
IQR = Q3 - Q1
lower, upper = Q1 - 1.5 * IQR, Q3 + 1.5 * IQR

final_features_filtered = final_features[
    final_features['ASL_Score'].between(lower, upper)].copy()

print(f"Data shape before: {final_features.shape}")
print(f"Data shape after outlier removal: {final_features_filtered.shape}")

X_filtered = final_features_filtered[selected_features]                    # Prepare Data
y_filtered = final_features_filtered['ASL_Score']

X_train_f, X_test_f, y_train_f, y_test_f = train_test_split(
    X_filtered, y_filtered, test_size=0.4, random_state=42)                 # Split & scale

scaler = StandardScaler()
X_train_f = scaler.fit_transform(X_train_f)
X_test_f = scaler.transform(X_test_f)

Lr_op = LinearRegression()                                       # Train & Evaluate Linear Regression
Lr_op.fit(X_train_f, y_train_f)
y_pred_f = Lr_op.predict(X_test_f)

Lr_op_mae = mean_absolute_error(y_test_f, y_pred_f)
Lr_op_mse = mean_squared_error(y_test_f, y_pred_f)
Lr_op_rmse = np.sqrt(Lr_op_mse)
Lr_op_R2 = r2_score(y_test_f, y_pred_f)

print(f"\nLinear Regression after Optimisation:")
print(f"  MAE: {Lr_op_mae:.2f}")
print(f"  MSE: {Lr_op_mse:.2f}")

print(f"  RMSE: {Lr_op_rmse:.2f}")

print(f"  R²:  {Lr_op_R2:.2f}")
```



```
Random Forest Feature Importances

                    Feature  Importance
6                 Child_Age    0.378945
2   Object touch (adult) Left    0.325182
3  Object touch (adult) Right    0.102949
0           EYE GAZE (adult)    0.090385
1           EYE GAZE (child)    0.045105
5  Object touch (child) Right    0.033157
4   Object touch (child) Left    0.013082
8               Voice_Coded    0.008855
7               Parent_Deaf    0.002339


Selected Features: ['Child_Age', 'Object touch (adult) Left', 'Object touch (adult) Right', 'EYE GAZE (adult)']
```

```
Data shape before: (24, 11)
Data shape after outlier removal: (23, 11)

Linear Regression after Optimisation:
  MAE: 63.18
  MSE: 5472.15
  RMSE: 73.97
  R²:  -0.71
<Figure size 640x480 with 0 Axes>
```

8 Hyperparameter Tuning for Random Forest

```
# Parameter Grid for Randomized Search
param_dist = {
    'n_estimators': [100, 200, 300, 400, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [10, 20, 30, 40, 50, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False] }

X_train_rf, X_test_rf, y_train_rf, y_test_rf = train_test_split(          # Data Split
    X, y, test_size=0.4, random_state=42)

rf = RandomForestRegressor(random_state=42)                              # Random Forest &
Randomized Search
random_search = RandomizedSearchCV(
    rf,
    param_distributions=param_dist,
    n_iter=50,                              # Lower for speed, higher for better search
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=-1
)

random_search.fit(X_train_rf, y_train_rf)                          # Fit Model

print("\nBest Parameters:", random_search.best_params_)            # Results
print("Best CV R² Score:", f"{random_search.best_score_:.4f}")

Rf_op = random_search.best_estimator_                              # Evaluation on Test Set
y_pred_best_rf = Rf_op.predict(X_test_rf)

Rf_op_mae = mean_absolute_error(y_test_rf, y_pred_best_rf)
Rf_op_mse = mean_squared_error(y_test_rf, y_pred_best_rf)
Rf_op_rmse = np.sqrt(Rf_op_mse)
Rf_op_R2 = r2_score(y_test_rf, y_pred_best_rf)

print("\nRandom Forest after Optimisation:")
print(f"  MAE: {Rf_op_mae:.2f}")
print(f"  MSE: {Rf_op_mse:.2f}")
print(f"  RMSE: {Rf_op_rmse:.2f}")
print(f"  R²:  {Rf_op_R2:.2f}")
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits

Best Parameters: {'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 4, 'max_features':
'sqrt', 'max_depth': 30, 'bootstrap': False}
Best CV R² Score: -0.6617
```

```
Random Forest after Optimisation:
  MAE: 37.75
  MSE: 1916.13
  RMSE: 43.77
  R²:  0.53
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_validation.py:528:
FitFailedWarning:
115 fits failed out of a total of 250.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
115 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_validation.py", line 866, in
_fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.11/dist-packages/sklearn/base.py", line 1382, in wrapper
    estimator._validate_params()
  File "/usr/local/lib/python3.11/dist-packages/sklearn/base.py", line 436, in _validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.11/dist-packages/sklearn/utils/_param_validation.py", line 98, in
validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' parameter of
RandomForestRegressor must be an int in the range [1, inf), a float in the range (0.0, 1.0], a str
among {'log2', 'sqrt'} or None. Got 'auto' instead.


  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One
or more of the test scores are non-finite: [        nan -2.28113442 -0.74941734         nan -0.70282971 -
0.70046085
```

```
 -2.35914688         nan -0.87236908         nan -1.49569467         nan
         nan         nan         nan -1.49405185         nan -1.39775911
         nan         nan         nan -0.97497989 -0.831945           nan
         nan -0.82627772 -0.66166348 -1.45692952 -2.50596003 -0.69137612
 -1.54369672         nan -0.82627772         nan         nan -1.49405185
 -2.17804411 -0.87236908 -0.70282971         nan -0.87968154         nan
         nan -1.54369672 -0.69874203 -0.74941734 -0.75117596         nan
         nan         nan]
  warnings.warn(
```

9 Optimisation of LSTM

```
data_path = '/content/drive/MyDrive/merged_data.csv'
merged_data = pd.read_csv(data_path)

# Feature columns
feature_cols = [
    'EYE GAZE (adult)', 'EYE GAZE (child)',
    'Object touch (adult) Left', 'Object touch (adult) Right',
    'Object touch (child) Left', 'Object touch (child) Right',
    'Child_Age', 'Parent_Deaf', 'Voice_Coded'
]

# Select features + target
data = merged_data[['Child ID'] + feature_cols + ['ASL_Activity']].copy()

# Clean numeric columns
for col in feature_cols:
    data[col] = pd.to_numeric(data[col], errors='coerce')
```

```
# Impute missing values
imputer = SimpleImputer(strategy='constant', fill_value=0)
data[feature_cols] = imputer.fit_transform(data[feature_cols])

# Scale features
scaler = StandardScaler()
data[feature_cols] = scaler.fit_transform(data[feature_cols])

# Convert to Sequence Data
sequence_data = []
sequence_targets = []
for _, group in data.groupby('Child ID'):
    X_seq = group[feature_cols].values
    y_seq = group['ASL_Activity'].values
    if len(X_seq) >= 10:
        sequence_data.append(X_seq)
        sequence_targets.append(np.mean(y_seq))

# Pad sequences
X_padded = pad_sequences(sequence_data, padding='post', dtype='float32')
y_array = np.array(sequence_targets, dtype='float32')

# Train-Test split
X_train_full, X_test, y_train_full, y_test = train_test_split(
    X_padded, y_array, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(
    X_train_full, y_train_full, test_size=0.2, random_state=42)

# Build LSTM Model Function for Tuning
def build_lstm_model(hp):
```

```
model = models.Sequential()

# Bidirectional LSTM
model.add(
    layers.Bidirectional(
        layers.LSTM(
            units=hp.Int('lstm_units', min_value=32, max_value=256, step=32),
            dropout=hp.Float('lstm_dropout', 0.0, 0.5, step=0.1),
            recurrent_dropout=hp.Float('lstm_recurrent_dropout', 0.0, 0.5, step=0.1),
            return_sequences=False
        ),
        input_shape=(X_train.shape[1], X_train.shape[2])
    )
)

# Dense hidden layer
model.add(
    layers.Dense(
        units=hp.Int('dense_units', min_value=16, max_value=128, step=16),
        activation='relu'
    )
)
# Output
model.add(layers.Dense(1))

# Compile
model.compile(
    optimizer=tf.keras.optimizers.Adam(
        learning_rate=hp.Float('learning_rate', min_value=1e-4, max_value=1e-2, sampling='LOG')
    ),
        loss='mse',
        metrics=['mae']
    )

    return model

# Set Up Random Search Tuner
tuner = RandomSearch(
    build_lstm_model,
    objective='val_mae',
    max_trials=5,
    executions_per_trial=1,
    directory='lstm_tuning',
    project_name='asl_play'
)

stop_early = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Run Tuning
tuner.search(
    X_train, y_train,
    epochs=20,
    validation_data=(X_val, y_val),
    callbacks=[stop_early],
    batch_size=32)

# Get Best Hyperparameters
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
print("\nBest Hyperparameters:")
print(f"LSTM Units: {best_hps.get('lstm_units')}")
print(f"LSTM Dropout: {best_hps.get('lstm_dropout')}")
print(f"LSTM Recurrent Dropout: {best_hps.get('lstm_recurrent_dropout')}")
print(f"Dense Units: {best_hps.get('dense_units')}")
print(f"Learning Rate: {best_hps.get('learning_rate')}")

# Train Final Model with Best Hyperparameters
Lstm_op = tuner.hypermodel.build(best_hps)
history = Lstm_op.fit(
    X_train_full, y_train_full,
    epochs=20,
    validation_data=(X_test, y_test),
    callbacks=[stop_early],
    batch_size=32)

# Evaluate on Test Set
y_pred_best = Lstm_op.predict(X_test).flatten()

Lstm_op_mae = mean_absolute_error(y_test, y_pred_best)
Lstm_op_mse = mean_squared_error(y_test, y_pred_best)
Lstm_op_rmse = np.sqrt(Lstm_op_mse)
Lstm_op_R2 = r2_score(y_test, y_pred_best)

print("\nLSTM after Optimisation:")
print(f"  MAE: {Lstm_op_mae:.2f}")
print(f"  MSE: {Lstm_op_mse:.2f}")
print(f"  RMSE: {Lstm_op_rmse:.2f}")
print(f"  R²:  {Lstm_op_R2:.2f}")
```

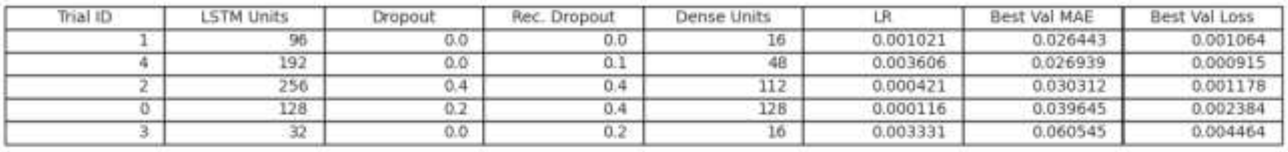
```
Trial 5 Complete [00h 02m 58s]
```

```
val_mae: 0.026938755065202713


Best val_mae So Far: 0.026443442329764366
Total elapsed time: 00h 10m 13s


Best Hyperparameters:
LSTM Units: 96
LSTM Dropout: 0.0
LSTM Recurrent Dropout: 0.0
Dense Units: 16
Learning Rate: 0.0010210396790978165
Epoch 1/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step - loss: 0.0060 - mae: 0.0618 -
val_loss: 8.0503e-04 - val_mae: 0.0211
Epoch 2/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - loss: 0.0017 - mae: 0.0327 -
val_loss: 8.2814e-04 - val_mae: 0.0226
Epoch 3/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - loss: 0.0020 - mae: 0.0337 -
val_loss: 9.8776e-04 - val_mae: 0.0239
Epoch 4/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - loss: 0.0018 - mae: 0.0299 -
val_loss: 8.4491e-04 - val_mae: 0.0243
Epoch 5/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - loss: 0.0011 - mae: 0.0274 -
val_loss: 4.9216e-04 - val_mae: 0.0185
Epoch 6/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - loss: 9.2400e-04 - mae: 0.0242 -
val_loss: 2.6418e-04 - val_mae: 0.0119
Epoch 7/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - loss: 0.0010 - mae: 0.0243 -
val_loss: 3.1120e-04 - val_mae: 0.0111
```

```
Epoch 8/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - loss: 9.8440e-04 - mae: 0.0245 - val_loss: 5.2008e-04 - val_mae: 0.0159
Epoch 9/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - loss: 8.3751e-04 - mae: 0.0238 - val_loss: 7.8330e-04 - val_mae: 0.0213
Epoch 10/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - loss: 7.0114e-04 - mae: 0.0217 - val_loss: 0.0010 - val_mae: 0.0262
Epoch 11/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - loss: 6.4691e-04 - mae: 0.0198 - val_loss: 0.0011 - val_mae: 0.0282
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 337ms/step

LSTM after Optimisation:
  MAE: 0.01
  MSE: 0.00
  RMSE: 0.02
  R²:  0.49
```

```python
# Collect results for all trials without retraining
trial_results = []

for trial_id, trial in tuner.oracle.trials.items():
    hp = trial.hyperparameters.values
    trial_results.append({
        'Trial ID': trial_id,
        'LSTM Units': hp.get('lstm_units'),
        'Dropout': hp.get('lstm_dropout'),
        'Rec. Dropout': hp.get('lstm_recurrent_dropout'),
        'Dense Units': hp.get('dense_units'),
        'LR': hp.get('learning_rate'),
        'Best Val MAE': trial.metrics.get_best_value('val_mae'),
        'Best Val Loss': trial.metrics.get_best_value('val_loss')
    })

# Convert to DataFrame
results_df = pd.DataFrame(trial_results)

# Sort by Best Val MAE (lower is better)
results_df = results_df.sort_values(by='Best Val MAE').reset_index(drop=True)

# Round values for cleaner table
results_df = results_df.round(6)

# Print DataFrame
print(results_df)

# Plot the table
fig, ax = plt.subplots(figsize=(12, len(results_df) * 0.5 + 2))
ax.axis('tight')
ax.axis('off')

table = ax.table(
    cellText=results_df.values,
    colLabels=results_df.columns,
    loc='center'
)

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

plt.show()
```

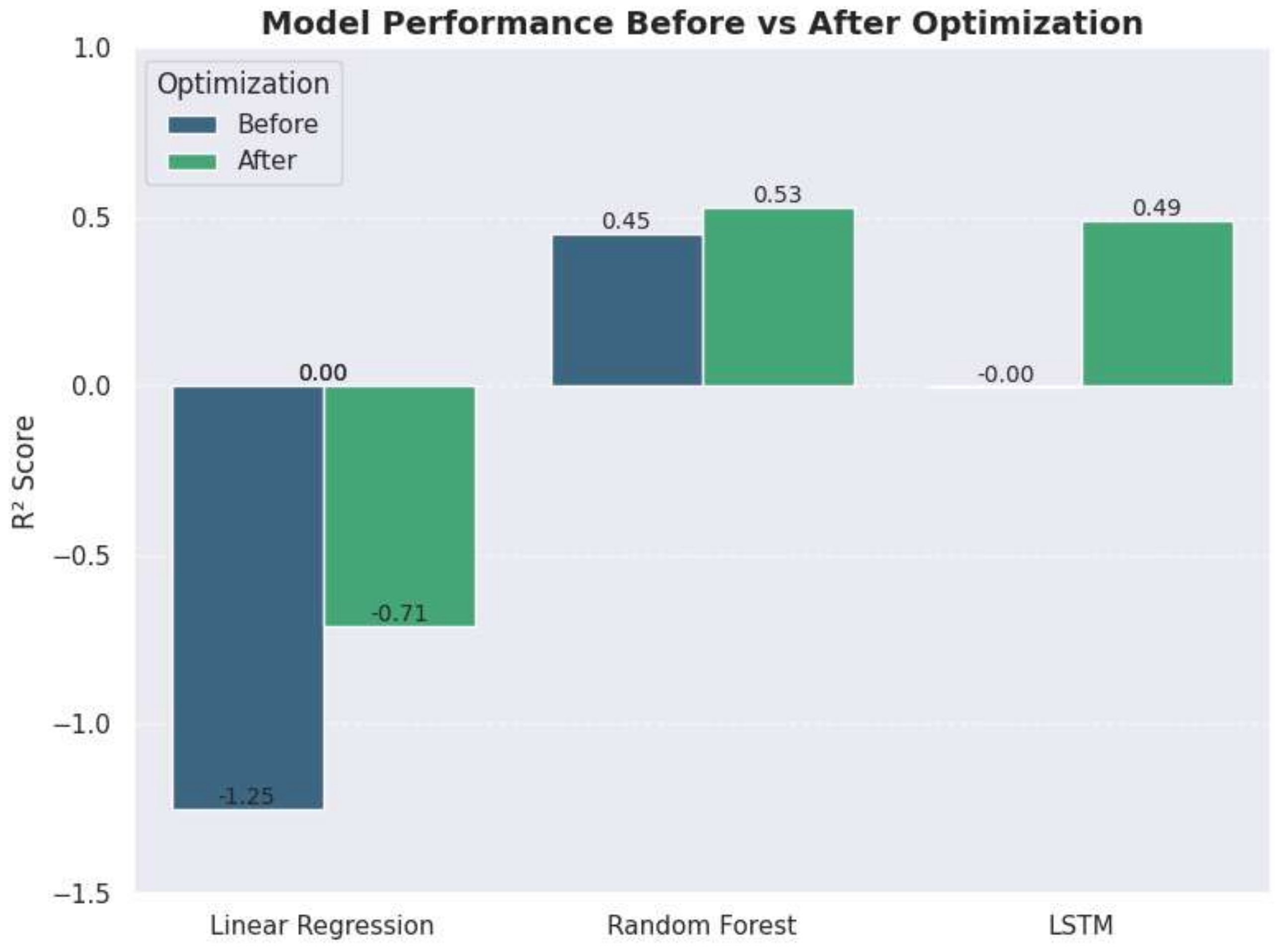
```
  Trial ID  LSTM Units  Dropout  Rec. Dropout  Dense Units        LR  \
0        1          96      0.0           0.0           16  0.001021
1        4         192      0.0           0.1           48  0.003606
2        2         256      0.4           0.4          112  0.000421
3        0         128      0.2           0.4          128  0.000116
4        3          32      0.0           0.2           16  0.003331

   Best Val MAE  Best Val Loss
0      0.026443       0.001064
1      0.026939       0.000915
2      0.030312       0.001178
3      0.039645       0.002384
4      0.060545       0.004464
```



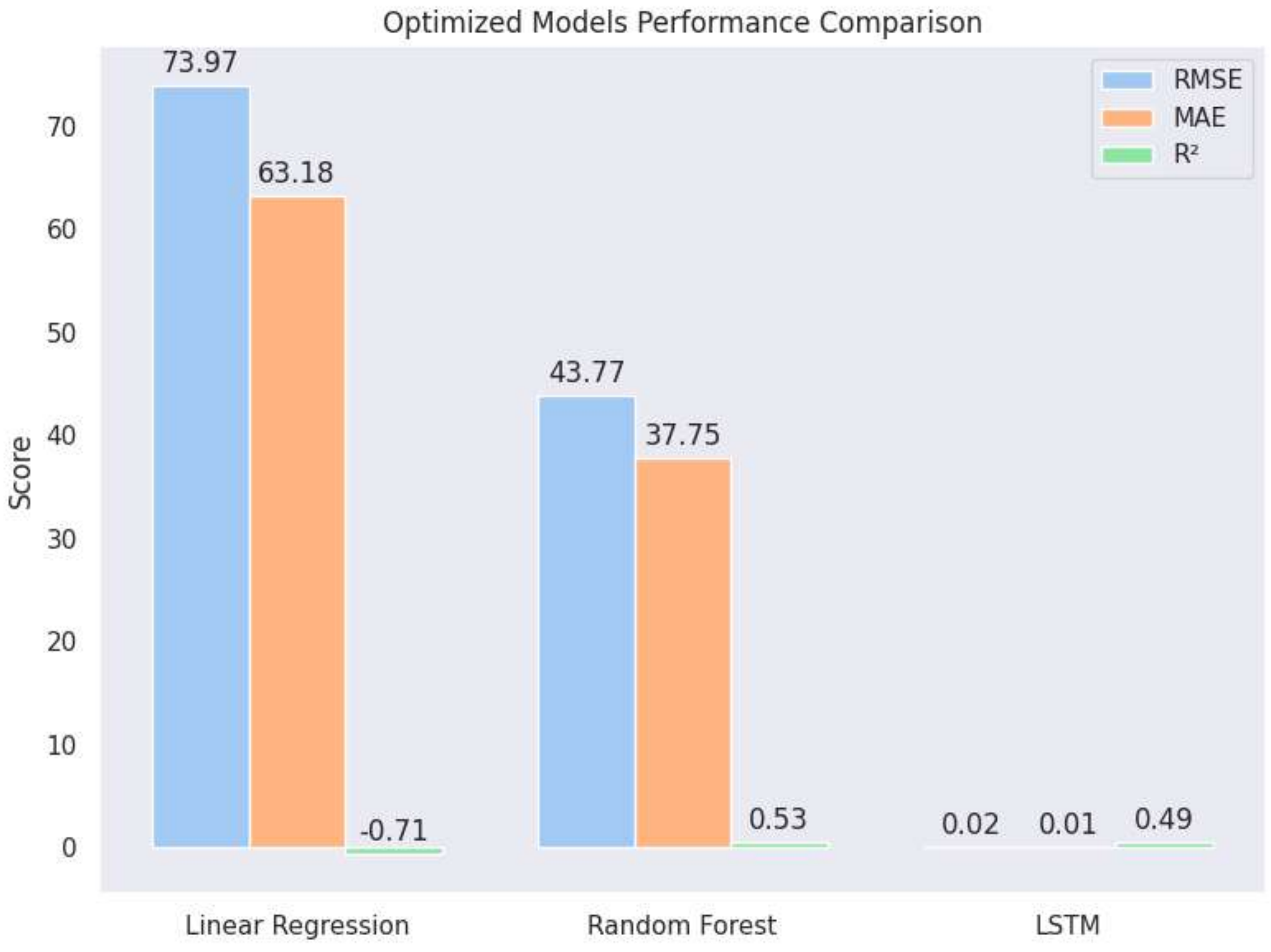
Trial ID	LSTM Units	Dropout	Rec. Dropout	Dense Units	LR	Best Val MAE	Best Val Loss
1	96	0.0	0.0	16	0.001021	0.026443	0.001064
4	192	0.0	0.1	48	0.003606	0.026939	0.000915
2	256	0.4	0.4	112	0.000421	0.030312	0.001178
0	128	0.2	0.4	128	0.000116	0.039645	0.002384
3	32	0.0	0.2	16	0.003331	0.060545	0.004464

10 Comparison of Model Performance (Before and After Optimization)

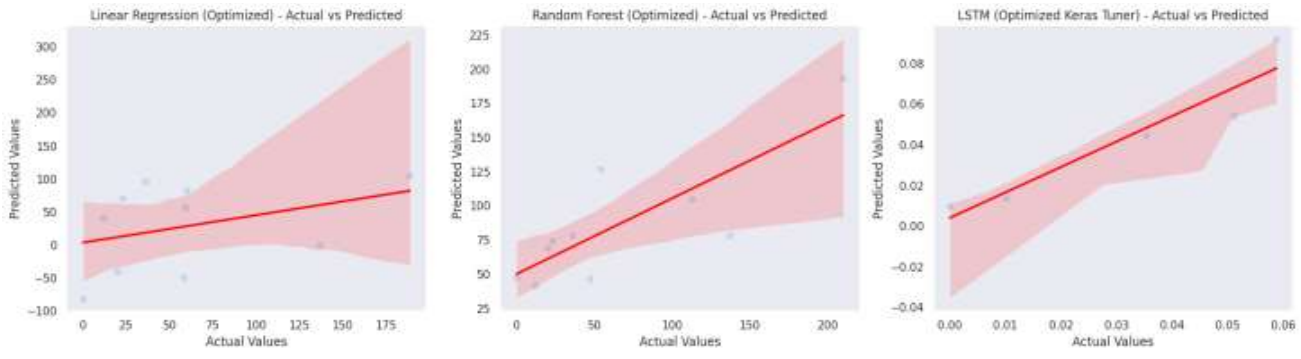
```
# R² scores
data = {
    'Model': ['Linear Regression', 'Linear Regression',
              'Random Forest', 'Random Forest',
              'LSTM', 'LSTM'],
    'Optimization': ['Before', 'After',
                     'Before', 'After',
                     'Before', 'After'],
    'R² Score': [Lr_R2, Lr_op_R2,
                 Rf_R2, Rf_op_R2,
                 Lstm_R2, Lstm_op_R2]
}

# Create DataFrame
df = pd.DataFrame(data)

# Plot
plt.figure(figsize=(8, 6))
sns.barplot(
    data=df,
    x='Model', y='R² Score', hue='Optimization',
    palette='viridis')

# Add score labels on bars
for p in plt.gca().patches:
    value = p.get_height()
    if pd.notnull(value):
        plt.gca().annotate(f"{value:.2f}",
                           (p.get_x() + p.get_width() / 2, value),
                           ha='center', va='bottom', fontsize=10)

plt.title('Model Performance Before vs After Optimization', fontsize=14, weight='bold')
plt.ylabel('R² Score')
plt.xlabel('')
plt.ylim(-1.5, 1.0)
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.legend(title='Optimization', loc='upper left')
plt.tight_layout()
plt.show()
```

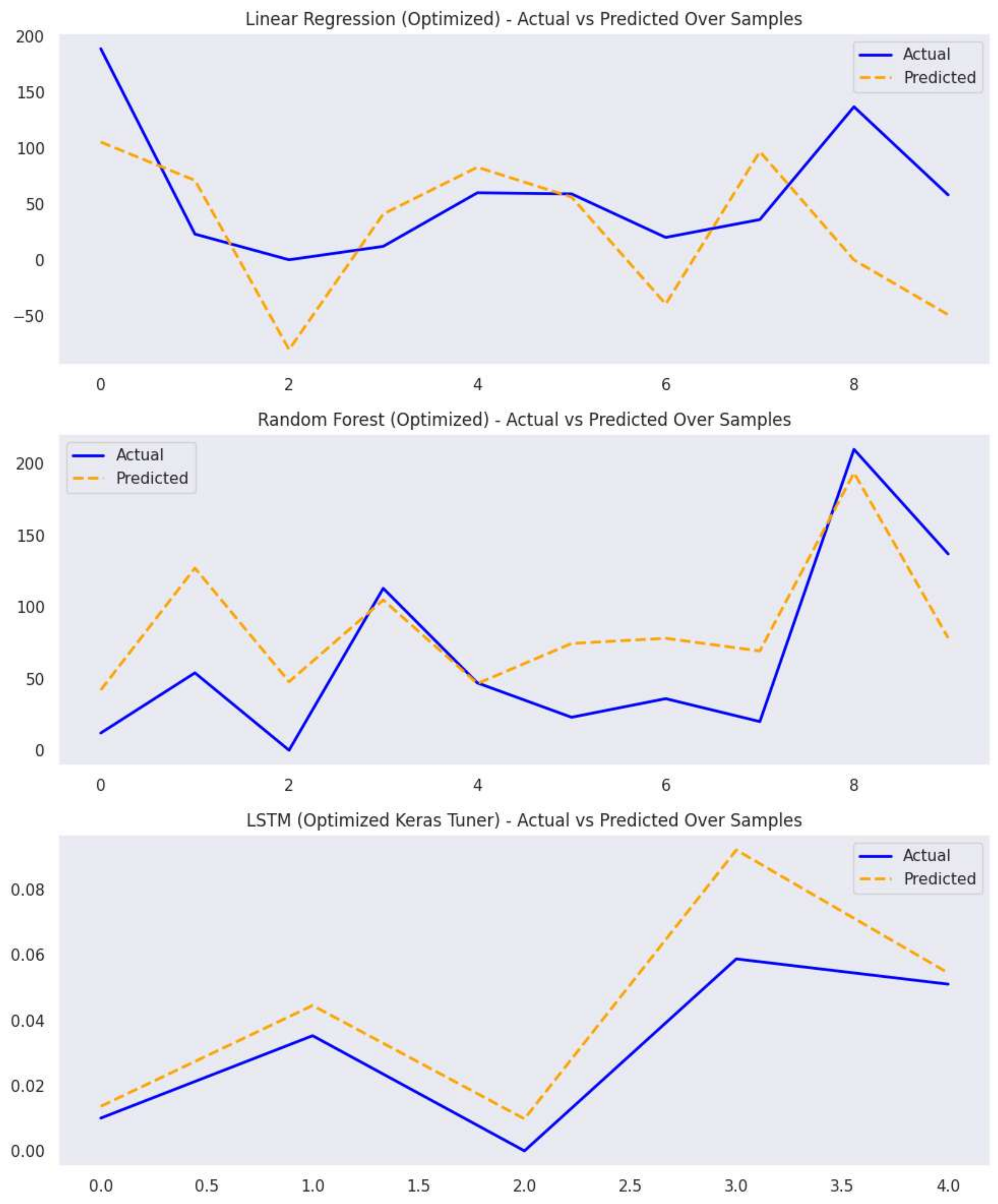


```
# Create dictionary directly from variables
model_scores = {
    "Linear Regression": (Lr_op_rmse, Lr_op_mae, Lr_op_R2),
    "Random Forest": (Rf_op_rmse, Rf_op_mae, Rf_op_R2),
    "LSTM": (Lstm_op_rmse, Lstm_op_mae, Lstm_op_R2) }

# Plotting
metrics = ["RMSE", "MAE", "R²"]
models = list(model_scores.keys())

rmse_values = [model_scores[m][0] for m in models]
mae_values  = [model_scores[m][1] for m in models]
r2_values   = [model_scores[m][2] for m in models]

x = np.arange(len(models))
bar_width = 0.25

fig, ax = plt.subplots(figsize=(8, 6))
bars1 = ax.bar(x - bar_width, rmse_values, width=bar_width, label='RMSE')
bars2 = ax.bar(x, mae_values, width=bar_width, label='MAE')
bars3 = ax.bar(x + bar_width, r2_values, width=bar_width, label='R²')

ax.set_ylabel('Score')
ax.set_title('Optimized Models Performance Comparison')
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()

# Annotate values
for bars in [bars1, bars2, bars3]:
    for bar in bars:
        height = bar.get_height()
        ax.annotate(f'{height:.2f}',
        xy=(bar.get_x() + bar.get_width() / 2, height),
        xytext=(0, 3),
        textcoords="offset points",
        ha='center', va='bottom')

plt.tight_layout()
plt.show()
```



```
# Create a list of tuples, each containing the model name, actual test values, and predicted values
models_to_plot = [
    ("Linear Regression (Optimized)", y_test_f, y_pred_f),
    ("Random Forest (Optimized)", y_test_rf, y_pred_best_rf),
    ("LSTM (Optimized)", y_test, y_pred_best) ]

# Correlation Scatter Plots (Actual vs Predicted)
fig, axes = plt.subplots(1, 3, figsize=(18, 5))
for ax, (model_name, y_actual, y_pred) in zip(axes, models_to_plot):
    sns.regplot(x=y_actual, y=y_pred, ax=ax, scatter_kws={'alpha':0.5}, line_kws={'color':'red'})
    ax.set_title(f"{model_name} - Actual vs Predicted")
    ax.set_xlabel("Actual Values")
    ax.set_ylabel("Predicted Values")
plt.tight_layout()
plt.show()

# Line Plots (Prediction vs Actual)
fig, axes = plt.subplots(3, 1, figsize=(10, 12))
for ax, (model_name, y_actual, y_pred) in zip(axes, models_to_plot):
    ax.plot(y_actual.values if isinstance(y_actual, pd.Series) else y_actual, label="Actual", color='blue', linewidth=2)
    ax.plot(y_pred, label="Predicted", color='orange', linestyle='--', linewidth=2)
    ax.set_title(f"{model_name} - Actual vs Predicted Over Samples")
    ax.legend()
plt.tight_layout()
plt.show()
```




Linear Regression (Optimized) - Actual vs Predicted Over Samples
Actual
Predicted
200
150
100
50
0
−50
0
2
4
6
8
Random Forest (Optimized) - Actual vs Predicted Over Samples
Actual
Predicted
200
150
100
50
0
0
2
4
6
8
LSTM (Optimized Keras Tuner) - Actual vs Predicted Over Samples
Actual
Predicted
0.08
0.06
0.04
0.02
0.00
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Unit X

10. References

- Anderson, D. & Reilly, J. (2002). 'The MacArthur Communicative Development Inventory: Normative Data for American Sign Language', *Journal of Deaf Studies and Deaf Education*, 7(2), pp. 83–106. (Available at: https://doi.org/10.1093/deafed/7.2.83)
- Baldwin, D.A. (1995) 'Understanding the link between joint attention and language', in Moore, C. & Dunham, P.J. (eds.) *Joint attention: Its origins and role in development*. Hillsdale, NJ: Lawrence Erlbaum Associates, pp.131–158.
- Bates, E. & Goodman, J. (1997) 'On the emergence of grammar from the lexicon', *Child Development*, 68(4), pp.789–804. (Available at: https://doi.org/10.2307/1132116)
- Bergstra, J. and Bengio, Y., 2012. *Random search for hyper-parameter optimization*. Journal of Machine Learning Research, 13(10), pp.281–305. (Available at Random Search for Hyper-Parameter Optimization)
- Breiman, L. (2001) 'Random forests', *Machine Learning*, 45(1), pp.5–32. (Available at: https://doi.org/10.1023/A:1010933404324)
- Camgoz, N.C., Koller, O., Hadfield, S. & Bowden, R. (2018) 'Neural Sign Language Translation', in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp.7784–7793. (Available at: https://doi.org/10.1109/CVPR.2018.00812)
- Cohen, J., Cohen, P., West, S.G. & Aiken, L.S. (2003) *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences*. 3rd edn. Mahwah, NJ: Lawrence Erlbaum Associates.
- Crasborn, O. & Sloetjes, H. (2008) 'Enhanced ELAN functionality for sign language corpora', in *Proceedings of the 3rd Workshop on the Representation and Processing of Sign Languages: Construction and Exploitation of Sign Language Corpora*. Paris: European Language Resources Association (ELRA), pp.39–43.
- Dupoux, E. (2016) 'Cognitive science perspectives on early language acquisition', *Trends in Cognitive Sciences*, 20(7), pp.529–541.
- Gelman, A. & Hill, J. (2007) *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge: Cambridge University Press.
- Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R. & Schmidhuber, J. (2017) 'LSTM: A Search Space Odyssey', *IEEE Transactions on Neural Networks and Learning Systems*, 28(10), pp.2222–2232. (Available at: https://doi.org/10.1109/TNNLS.2016.2582924)
- Hochreiter, S. & Schmidhuber, J. (1997) 'Long short-term memory', *Neural Computation*, 9(8), pp.1735–1780. (Available at: https://doi.org/10.1162/neco.1997.9.8.1735)

- Kohavi, R. (1995) ‘A study of cross-validation and bootstrap for accuracy estimation and model selection’, in *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pp.1137–1143.

- Koller, O., Camgoz, N.C., Ney, H. and Bowden, R. (2019). Weakly supervised learning with multi-stream CNN-LSTM-HMMs to discover sequential parallelism in sign language videos. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(9), pp.2306–2320.

- Kuhl, P.K. (2004) ‘Early language acquisition: cracking the speech code’, *Nature Reviews Neuroscience*, 5(11), pp.831–843. (Available at: https://doi.org/10.1038/nrn1533)

- Lieberman, A.M., Hatrak, M. & Mayberry, R.I. (2014) ‘Learning to look for language: Development of joint attention in young deaf children’, *Language Learning and Development*, 10(1), pp.19–35. (Available at: https://doi.org/10.1080/15475441.2012.760381)

- Lieberman, A.M., Hatrak, M. & Mayberry, R.I. (2022) ‘ASL-PLAY: A dataset of early American Sign Language acquisition’, in *Proceedings of the 47th Annual Boston University Conference on Language Development*. Somerville, MA: Cascadilla Press.

- Lieberman, A., Gappmayr, P. & Fitch, A. (2022) *ASL-PLAY: A dataset of early American Sign Language acquisition* [dataset]. Open Science Framework. Available at: https://osf.io/3w8ka/ (Accessed: 9 July 2025).

- Lipnevich, A.A. & Panadero, E. (2021). A review of feedback models and theories: Descriptions, definitions, and conclusions. *Frontiers in Education*, 6, Article 720195.

- Lipton, Z.C., Berkowitz, J. & Elkan, C. (2015) ‘A critical review of recurrent neural networks for sequence learning’, *arXiv preprint arXiv:1506.00019*. (Available at: https://arxiv.org/abs/1506.00019)

- Lundberg, S.M. & Lee, S.I. (2017) ‘A Unified Approach to Interpreting Model Predictions’, in *Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS)*, pp.4765–4774. (Available at: https://papers.nips.cc/paper_files/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html)

- MacCann, C., Jiang, Y., Brown, L.E.R., Double, K.S., Bucich, M. and Minbashian, A. (2020). Emotional intelligence predicts academic performance: A meta-analysis. *Psychological Bulletin*, 146(2), pp.150–186. (Available at: Emotional Intelligence Predicts Academic Performance)

- Mayberry, R.I. & Lock, E. (2003) ‘Age constraints on first versus second language acquisition: Evidence for linguistic plasticity and epigenesis’, *Brain and Language*, 87(3), pp.369–384. (Available at: https://doi.org/10.1016/S0093-934X(03)00137-8)

- Meier, R.P. (1991) ‘The acquisition of American Sign Language’, in *Language Acquisition by Eye*. Hillsdale, NJ: Lawrence Erlbaum Associates.

- Orlansky, M. D., & Bonvillian, J. D. (1985). Sign language acquisition: Language development in children of deaf parents and implications for other populations. *Merrill-Palmer Quarterly, 31*(2), 127–143.

- Tomasello, M. (1995). 'Joint attention as social cognition'. In: Moore, C. & Dunham, P.J. (eds.) *Joint Attention: Its Origins and Role in Development*. Hillsdale, NJ: Lawrence Erlbaum Associates, pp. 103–130.

- Tomasello, M. (2003) *Constructing a language: A usage-based theory of language acquisition*. Cambridge, MA: Harvard University Press.

- Wolf, T. et al. (2020) 'Transformers: State-of-the-Art Natural Language Processing', in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. (Available at: https://aclanthology.org/2020.emnlp-demos.6)

The End