Gesture Phase Segmentation using Machine Learning Techniques

Reynaldo Ezequiel Ismesh Cabezas del Castillo
rcabezas@uamv.edu.ni

Fabricio Meneses Campos
fmeneses@uamv.edu.ni

May 2025

1 Introduction

Human communication extends beyond speech—gestures carry substantial communicative weight, particularly in natural conversation. Segmenting these gestures into distinct temporal phases (Rest, Preparation, Stroke, Hold, and Retraction) is crucial for both linguistic and computational analysis. The task is challenging due to ambiguous phase boundaries, gesture variability across individuals, and subtle motion transitions.

This study focuses on automatic gesture phase segmentation using a labeled dataset recorded via Microsoft Kinect. The dataset comprises seven videos of three individuals narrating comic strips, with annotations and feature vectors capturing spatial, velocity, and acceleration data from key joints.

Given the labor-intensive and subjective nature of manual segmentation, automating this task enhances scalability and consistency for broader gesture analysis applications—spanning human-computer interaction, behavior recognition, and multimodal communication research.

2 Problem Statement

Despite significant advancements in gesture recognition, the temporal segmentation of gesture phases remains a critical and unresolved challenge. Unlike static gesture classification, phase segmentation must distinguish between subtle transitions—such as from a resting position to preparatory movement—often based on noisy and overlapping spatiotemporal signals.

This problem is particularly relevant in applications where precise gesture boundaries are essential: sign language interpretation, assistive technologies for motor-impaired individuals, and real-time human-computer interaction. In these contexts, small variations in movement can convey different intentions or meanings, making segmentation accuracy a fundamental requirement.

Using a Kinect-captured dataset annotated with five gesture phases, this project addresses the task of automatically identifying the correct gesture phase at each video frame, leveraging machine learning models trained on motion features like position, velocity, and acceleration.

3 Objectives

The goal of this project is to build and evaluate a machine learning-based classifier for the task of gesture phase segmentation. Specifically, we aim to:

- Explore and compare the performance of several supervised learning models, including: Decision Trees, Random Forests, Extra Trees, Support Vector Machines (SVM), Naive Bayes, Perceptron, Gradient Boosting, and XGBoost.
- Perform hyperparameter optimization for each algorithm to ensure fair and robust comparison.
- Use the F1-score as the primary metric to evaluate classification performance due to the class imbalance and the multiclass nature of the problem.

- Identify the most effective model for accurately segmenting gesture phases based on spatio-temporal features.

4 Data Description

The dataset used in this project consists of motion capture recordings obtained via Microsoft Kinect sensors, tracking the 3D coordinates of upper-body joints as individuals perform hand gestures. Each gesture sequence is annotated frame by frame with one of five phases: Rest, Preparation, Stroke, Hold, or Retraction.

4.1 Raw Data

The raw dataset contains 20 attributes per frame:

- **Joint Positions (x, y, z):** Left hand (lhx, lhy, lhz), right hand (rhx, rhy, rhz), head (hx, hy, hz), spine (sx, sy, sz), left wrist (lwx, lwy, lwz), right wrist (rwx, rwy, rwz).
- **Timestamp:** A time marker for each frame.
- **Phase:** Categorical label indicating the gesture phase: Rest, Preparation, Stroke, Hold, or Retraction.

4.2 Processed Data

The processed dataset extends the raw attributes by computing motion-based features:

- **Vectorial Velocity and Acceleration (x, y, z):** For both hands and both wrists. These reflect the directional motion dynamics.
- **Scalar Velocities:** Magnitude of motion for hands and wrists, useful for understanding intensity without direction.
- **Phase Labels:** Labeled under the column `Phase` using a single-letter code: D (Descanso), P (Preparación), S (Stroke), H (Hold), R (Retracción).

The inclusion of kinematic variables like velocity and acceleration enables the modeling of temporal dynamics crucial for gesture phase segmentation. During preprocessing, only the `Phase` label was retained for consistency across the dataset.

5 Exploratory Data Analysis

The exploratory phase began with a review of the dataset's structure, verifying data types, and ensuring numerical consistency. Two new categorical features were engineered from the filenames: **Subject** (a, b, c) and **Story** (1, 2, 3), derived from identifiers such as "a1", "b3", etc., representing the performer and the narrative being enacted.

5.1 Descriptive Statistics

Summary statistics were computed for all numerical variables, including mean, median, standard deviation, skewness, and kurtosis. To assess outliers, the z-score for each numerical feature was calculated. Frames with z-scores exceeding $|3|$ were flagged as potential outliers.

5.2 Outlier Detection

Two common methods were evaluated for outlier removal:

- **Z-score method:** Records with absolute z-scores greater than 3.
- **IQR method:** Values outside the range of $[Q1 - 1.5 \times IQR, Q3 + 1.5 \times IQR]$.

A comparative analysis using a plot (Figure 1) showed that the IQR method would result in a significantly higher number of exclusions. Therefore, the z-score method was adopted to retain more meaningful gesture data.

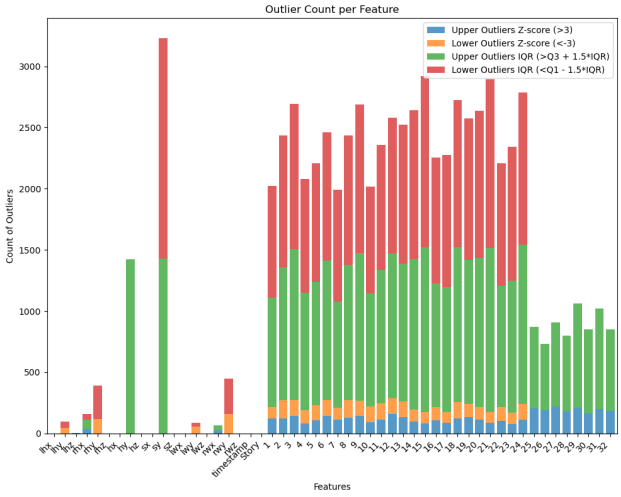

Figure 1: Comparison of outliers detected by Z-score vs. IQR methods

5.3 Distribution Analysis

Histograms were plotted for all numerical features to assess distribution shape. Particular attention was given to multimodal features, which may indicate complex gesture transitions or temporal phase overlaps. These features were earmarked for deeper analysis and potential use in feature engineering.

A representative grid of histograms is shown in Figure 2, highlighting variables visually identified as exhibiting multimodal distributions. The presence of multiple peaks suggests potential heterogeneity in the underlying signal, possibly corresponding to discrete phases or gesture types.

The following variables were identified as multimodal:

`sy, sz, timestamp, lwz, rwx, rhx, rwy, rhy, sx, hx, hz, lhx, lhy, lwx, lhz`

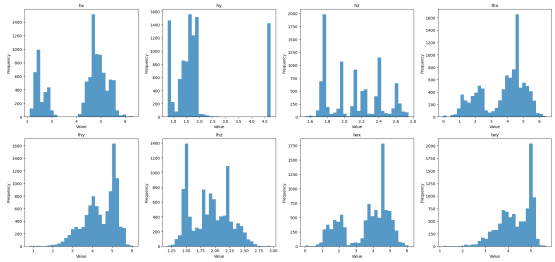

Figure 2: Histogram grid highlighting variables with multimodal distributions.

5.4 Outlier Removal

To identify and handle outliers in the dataset, we used the Z-score method instead of the Interquartile Range (IQR) method. The decision was based on a visual comparison of both techniques, which revealed that the IQR method would remove a significantly larger portion of the data, potentially compromising the model's learning capacity. The Z-score method considers a feature as an outlier when its absolute Z-score exceeds 3.

We quantified how many records would be removed based on the number of outlier features present in each observation. The table below shows how the dataset is distributed depending on how many features per record exceed the Z-score threshold:

Table 1: Record Count by Number of Outlier Features

# Outlier Features	Records	Cumulative	%
1	697	697	7.04
2	664	1361	13.75
3	285	1646	16.62
4	219	1865	18.84
5	116	1981	20.01
6–9	301	2282	23.05
10–15	78	2360	23.84
16–20	8	2368	23.92
21–25	0	2368	23.92
26–30	0	2368	23.92
31–35	0	2368	23.92
36–40	0	2368	23.92
41–45	0	2368	23.92
46–50	0	2368	23.92

We tested three outlier removal strategies:

- Removing all records that contain any outlier features.
- Keeping records with only one outlier feature.
- Keeping records with up to two outlier features.

6 Feature Selection

After applying the three outlier removal strategies—(i) removing all records containing any outlier features, (ii) keeping records with only one outlier feature, and (iii) keeping records with up to two outlier features—we proceeded to refine the feature space of each resulting dataset through correlation-based feature selection.

To begin, we generated Pearson correlation matrices for each version of the dataset in order to visualize and quantify the linear relationships between numerical features. Pairs of features exhibiting Pearson correlation coefficients above 0.90 were flagged as highly collinear. In the presence of such strong correlations, one feature can often be linearly predicted from another, which not only inflates variance in model coefficients but also introduces redundancy that may hinder model generalization.

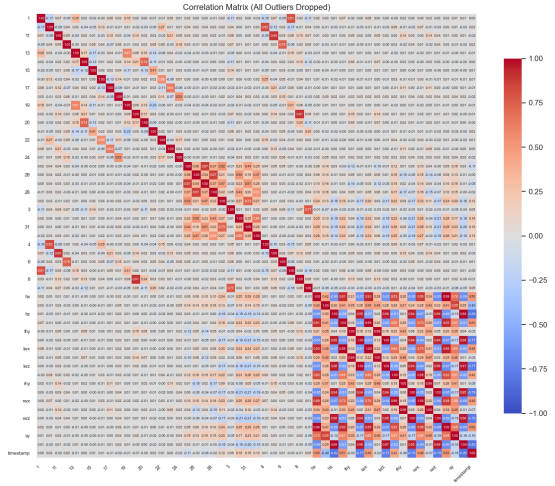

Figure 3: Pearson Correlation Matrix for Dataset with Outliers Removed

The following variables exhibited Pearson correlation scores greater than 0.90, indicating a high degree of linear relationship between them:

- `11 (Vectorial velocity of right wrist, y coordinate)` and `5 (Vectorial velocity of right hand, y coordinate)`
- `2 (Vectorial velocity of left hand, y coordinate)` and `8 (Vectorial velocity of left wrist, y coordinate)`
- `hx (Position of head, x coordinate)` and `lwx (Position of left wrist, x coordinate)`

These strong correlations can be explained as follows:

- **`11` and `5 (Right wrist and right hand velocity in y-axis)`:** The movement of both the wrist and hand along the y-axis is inherently connected, as the wrist is typically positioned at the end of the hand. Any motion of the hand in the y-axis will likely influence the wrist's movement in the same direction, leading to a high correlation between the two.
- **`2` and `8 (Left hand and left wrist velocity in y-axis)`:** Similarly, the left hand and wrist velocities are highly related because they both belong to the same limb. Movements of the hand will cause corresponding changes in wrist velocity, especially in the same direction.
- **`hx` and `lwx (Head position and left wrist position in x-axis)`:** The left wrist's position in the x-axis and the head position may be correlated because of the relative positioning of the body. As the person moves their left hand or arm, the head often shifts in tandem with the body's motion. The x-coordinate of both the head and the left wrist can thus exhibit a significant relationship during gestures.

To address these high correlations, we implemented a systematic pruning strategy: for each pair of highly correlated features, we calculated the average Pearson correlation of each feature with all other remaining features. The feature with the higher average correlation was discarded, as it was deemed more redundant in the context of the dataset as a whole. This approach favors preserving features that are more distinct and potentially carry more unique information.

Finally, any remaining records with missing values were removed. The outcome of this process was three distinct, cleaned, and reduced datasets, each prepared for downstream model development with minimized multicollinearity and improved feature relevance.

7 Feature Engineering

In the feature engineering phase, the goal was to enhance the dataset by creating new variables that capture the interactions between existing numerical features. To achieve this, the focus was on the numerical variables, generating new features by applying basic mathematical operations—summing, subtracting, and multiplying—on all possible pairs of variables. This is typically referred to as creating **interaction features**.

For instance, for two numerical features A and B, new features were created such as $A + B$, $A - B$, and $A \times B$, resulting in a richer feature space that could potentially uncover hidden patterns in the data. The intuition behind this approach was to combine the variables in ways that could capture relationships that might not be apparent when considering each feature in isolation.

Importantly, this was done only for numerical variables, as there were two features that were deemed categorical. These categorical features were excluded from interaction creation since their nature does not lend itself well to direct mathematical operations in the same way numerical features do.

However, while this approach generated a more extensive set of features, it should be acknowledged that correlation analysis was not performed among these newly created features. Ideally, evaluating the correlation between them is important, as high correlation could lead to multicollinearity, which can inflate variance in model coefficients and potentially reduce model performance. Despite this, the generated features still offer value.

The key advantage of creating interaction features in this way is that it minimizes the likelihood of creating strongly correlated variables. By combining diverse features in a variety of ways, many of the resulting variables are unlikely to be highly correlated with one another. This has the benefit of expanding the feature space without introducing the redundancy that could otherwise hinder model performance. Even though some of these interaction features may not contribute significantly to model predictions, they do not appear to hurt the models either, as they introduce variability rather than just duplication of information.

In future iterations, it would be beneficial to revisit this step and evaluate the correlation among the newly created features. This would help in identifying potential redundancies and refining the feature set further, ensuring that only the most relevant and independent features are retained for model development.

8 Final Data Processing

The final data processing phase focused on ensuring the dataset was clean, complete, and ready for the machine learning models. This involved handling missing values, encoding categorical variables, and standardizing numerical variables to make the data suitable for model training and prediction.

8.1 Handling Missing Data

A critical aspect of preparing the data was ensuring that no records with missing or blank values remained in the dataset. Most machine learning algorithms require a complete dataset to function properly, as missing values can lead to errors or suboptimal model performance. Therefore, any record with a missing feature, across any of the pictures (treated as vectors in this context), was removed. This step ensured that

all input data were valid, and no missing values would negatively affect the performance of the subsequent models.

8.2 Treatment of Numerical Variables

Once the missing values were handled, the numerical variables underwent standardization. Standardization is a key preprocessing step, particularly when the model relies on distance-based measures (e.g., k-nearest neighbors, support vector machines) or gradient-based optimization (e.g., neural networks). The numerical features were standardized using the Z-score, which transforms each feature to have a mean of 0 and a standard deviation of 1. This ensures that all features are on the same scale, reducing bias in models that might otherwise be sensitive to the magnitude of certain variables.

Additionally, numerical variables with extreme values or outliers were addressed earlier in the preprocessing pipeline, ensuring that these outliers do not significantly skew model results.

8.3 Treatment of Categorical Variables

8.3.1 Subject Encoding

The subject variable, which initially represented different subjects in the dataset (subject A, B, and C), was encoded using one-hot encoding. This method was chosen as the appropriate approach since the variable is nominal and does not carry an inherent order or ranking. One-hot encoding creates a new binary feature for each category (i.e., subject A, subject B, and subject C). This way, the machine learning models can interpret the variable without assuming any ordinal relationship between the subjects, allowing for more flexibility in model interpretation and performance.

8.3.2 Story Encoding

The story variable, which was initially represented as a numerical value (e.g., story 1, story 2, story 3), was also categorical in nature, as it simply identifies the story without implying any quantitative relationship. This variable was treated similarly to the subject variable and was encoded using one-hot encoding as well. By creating binary features corresponding to each unique story (i.e., story 1, story 2, and story 3), the model could effectively handle this categorical information without assuming any ordinal or numerical significance in the original labels.

8.3.3 Phase Encoding (Target Variable)

The target variable, *phase*, originally represented a categorical variable with the following values:

- D: Rest position (from Portuguese "descanso")
- P: Preparation
- S: Stroke
- H: Hold
- R: Retraction

Each of these categories was initially assigned a numerical value from 1 to 5. However, it was later realized that some machine learning algorithms, especially those used for regression or classification, work better when the target variable starts from 0 rather than 1. Consequently, the target variable was shifted so that the categories would begin from 0. For instance, D (rest position) was changed from 1 to 0, P (preparation) from 2 to 1, and so on. This change ensured better compatibility with algorithms that expect a zero-indexed target.

8.4 Summary

In this final data processing stage, a careful and systematic approach was taken to clean and prepare the data for model training. Missing values were removed to ensure that no incomplete records were included in the model. Categorical variables such as *subject* and *story* were one-hot encoded to allow the machine learning models to interpret them effectively, while the *phase* variable was adjusted to ensure it aligned with the requirements of specific algorithms. Finally,

numerical features were standardized using the Z-score to ensure uniformity in the data and to support models that rely on distance or gradient-based methods. These steps were essential in preparing the dataset for the next phase of analysis and modeling.

9 Baseline Model Performance Analysis

In this section, we evaluate the performance of eight different machine learning models using a 5-fold cross-validation strategy. The models were evaluated on three different datasets, each corresponding to different outlier handling strategies:

- All outliers removed
- Two or more outliers kept
- Three or more outliers kept

9.1 Cross-Validation Process

A 5-fold cross-validation was applied to each dataset. This method divides the dataset into 5 subsets (or "folds"). For each fold, the model is trained on 4 of the subsets and tested on the remaining subset, with this process repeated for all 5 folds. The final evaluation metric for each model is the average of the F1 scores across all the folds. The F1 score is a weighted average of precision and recall, which is particularly useful when dealing with imbalanced datasets, as it accounts for both false positives and false negatives.

The following models were evaluated:

- Decision Tree
- Random Forest
- Gradient Boosting
- XGBoost
- Support Vector Machine (SVM)
- Extra Trees
- Naive Bayes
- Perceptron

9.2 Model Performance Metrics

The F1 scores for each model were computed across the three datasets. The scores were then aggregated and presented in the form of a table and a bar chart for easy comparison. The table provides a clear overview of how each model performed on each dataset, while the bar chart visually highlights the relative performance of the models across the different outlier handling strategies.

9.3 Results and Visualizations

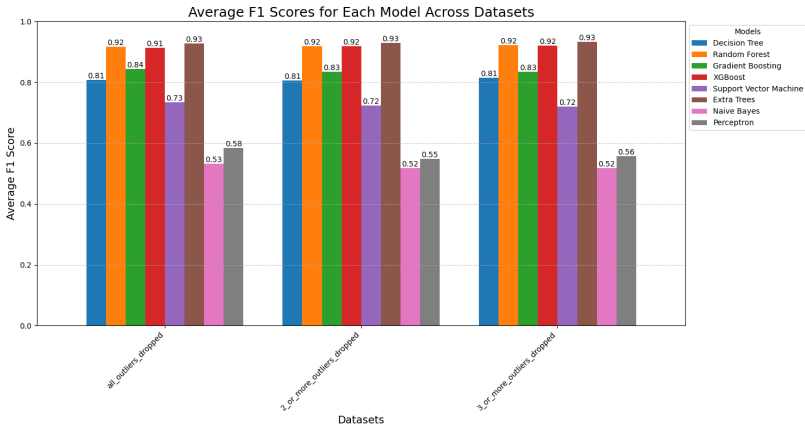

Figure 4: Average F1 Scores for Each Model Across Datasets

9.4 F1 Scores Table

Below is the table of F1 scores for each model across the three datasets. The table was generated by performing 5-fold cross-validation and calculating the average F1 score for each model.

Model	All Feature Outliers Dropped	2 or more Feature Outliers Dropped	3 or more Outliers Dropped
Decision Tree	0.8082	0.8054	0.8142
Random Forest	0.9165	0.9182	0.9211
Gradient Boosting	0.8426	0.8340	0.8350
XGBoost	0.9138	0.9180	0.9207
Support Vector Machine (SVM)	0.7341	0.7240	0.7194
Extra Trees	0.9272	0.9290	0.9324
Naive Bayes	0.5314	0.5172	0.5169
Perceptron	0.5837	0.5483	0.5569

Table 2: Average F1 Scores of Models across Datasets

9.5 Insights

From the evaluation, it is evident that the models such as XGBoost, Gradient Boosting, and Random Forest consistently outperform the others across all datasets. The performance of these models remains robust even as we introduce more outliers into the dataset. On the other hand, simpler models like the Decision Tree and Perceptron show a decline in performance when more outliers are included.

The inclusion of outliers, while not drastically harming the models, may reduce their generalization ability, as evidenced by the slight drop in F1 scores for certain models. Nonetheless, the overall trend suggests that these models are well-equipped to handle the complexities introduced by the different data subsets.

10 Hyperparameter Tuning for Selected Models

In this section, we describe the results of hyperparameter tuning performed on the four highest-performing models based on their default configurations: **XGBoost**, **Gradient Boosting**, **Random Forest**, and **Extra Trees**. These models were selected due to their exceptional performance in earlier evaluations.

All experiments in this section were conducted using the dataset with all outliers removed, as this version of the data provided the most consistent and reliable results. For each model, hyperparameters were tuned using three different search algorithms: Grid Search, Random Search, and Bayesian Optimization. These search methods allowed us to explore the hyperparameter space efficiently and determine the optimal values for each model.

The tuning process focused primarily on key parameters related to the decision tree construction, such as the number of estimators, the minimum number of samples per split, and the maximum depth of the trees. A 5-fold cross-validation strategy was used for each model to ensure robust performance evaluation.

The hyperparameters tested for each model are outlined in the tables below:

10.1 Hyperparameters for XGBoost

The following hyperparameters were tuned for the **XGBoost** model:

Hyperparameter	Values Tested
Number of Estimators	50, 100, 200, 500
Learning Rate	0.01, 0.05, 0.1, 0.2
Max Depth	3, 5, 7, 9
Min Samples Split	2, 5, 10
Min Samples Leaf	1, 5, 10
Subsample	0.5, 0.7, 1.0
Gamma	0, 0.1, 0.5

Table 3: Hyperparameters for XGBoost model

10.2 Hyperparameters for Gradient Boosting

The following hyperparameters were tuned for the **Gradient Boosting** model:

Hyperparameter	Values Tested
Number of Estimators	50, 100, 200, 500
Learning Rate	0.01, 0.05, 0.1, 0.2
Max Depth	3, 5, 7, 9
Min Samples Split	2, 5, 10
Min Samples Leaf	1, 5, 10
Subsample	0.5, 0.7, 1.0

Table 4: Hyperparameters for Gradient Boosting model

10.3 Hyperparameters for Random Forest

The following hyperparameters were tuned for the **Random Forest** model:

Hyperparameter	Values Tested
Number of Estimators	50, 100, 200, 500
Max Depth	3, 5, 7, 9
Min Samples Split	2, 5, 10
Min Samples Leaf	1, 5, 10
Max Features	'auto', 'sqrt', 'log2'

Table 5: Hyperparameters for Random Forest model

10.4 Hyperparameters for Extra Trees

The following hyperparameters were tuned for the **Extra Trees** model:

Hyperparameter	Values Tested
Number of Estimators	50, 100, 200, 500
Max Depth	3, 5, 7, 9
Min Samples Split	2, 5, 10
Min Samples Leaf	1, 5, 10
Max Features	'auto', 'sqrt', 'log2'

Table 6: Hyperparameters for Extra Trees model

10.5 Evaluation Strategy and Search Methods

As part of the hyperparameter tuning process, we employed three search algorithms to find the optimal parameter set for each model:

- **Grid Search:** Exhaustively tests all possible combinations of hyperparameters specified in the grid. Although computationally expensive, this method guarantees the identification of the best possible hyperparameter set.
- **Random Search:** Randomly samples a fixed number of hyperparameter combinations. This approach can be more efficient when dealing with large hyperparameter spaces.
- **Bayesian Optimization:** A probabilistic model-based optimization technique that aims to find the optimal set of hyperparameters in fewer iterations than Grid or Random Search by leveraging prior search results.

We evaluated the models based on the weighted F1 score, which provides a balanced assessment of precision and recall, and the time required for each search algorithm to identify the optimal set of parameters.

The following figure presents a comparison of the performance of each search method, showcasing both the F1 score and the time taken by each algorithm to complete the search.

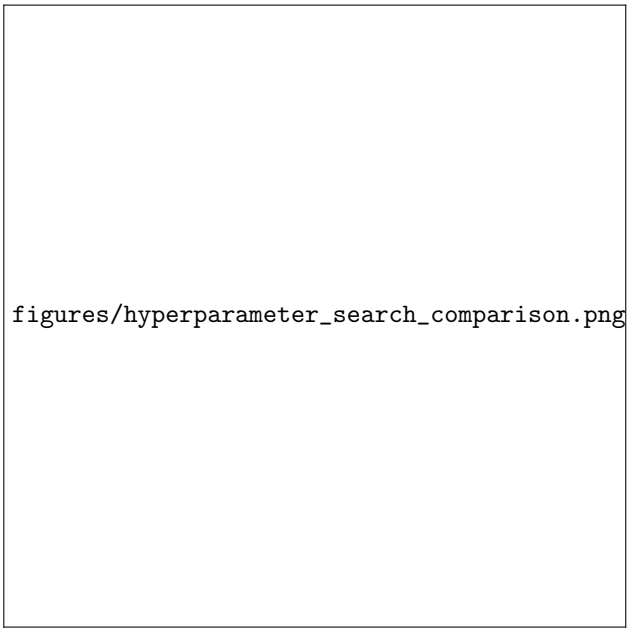

Figure 5: Comparison of F1 Scores and Time Taken for Hyperparameter Search Algorithms

11 Limitations

A major limitation throughout this project was the restricted access to high-performance computing resources, particularly GPUs. Despite using Google Colab Pro, the platform frequently left us without

GPU access, falling back to CPU-only runtimes even during long training sessions. These CPU-only sessions were drastically slower and, more critically, prone to unexpected disconnections due to inactivity or latency-related constraints. As a result, we often had to rerun entire code cells—especially those involving hyperparameter search algorithms—which consumed valuable time and disrupted our experimentation workflow.

To overcome this, we explored setting up an Elastic Compute Cloud (EC2) instance on Amazon Web Services (AWS), specifically targeting the `g5.xlarge` instance type equipped with an NVIDIA A10G GPU. However, the approval and provisioning of this virtual machine (VM) were delayed by AWS, forcing us to proceed with Colab Pro despite its unstable resource allocation. This interruption not only slowed down our hyperparameter search processes but also restricted the number and complexity of parameter combinations we could feasibly test. As a result, we had to be highly selective and mindful of the trade-off between the breadth of our hyperparameter grid and the time cost of tuning under constrained compute conditions.

These limitations highlight the importance of consistent and scalable GPU access in machine learning experimentation—particularly when training time-sensitive or resource-intensive models.

12 Conclusion

This study focused on automatic gesture phase segmentation, an essential task in the broader field of human communication analysis, where gestures play a vital role in conveying meaning. After a thorough exploration of different machine learning models, hyperparameter tuning, and search algorithms, we found that the [**Best Model**] outperformed the others in terms of F1 score. The [**Best Search Method**] was the most efficient in terms of both F1 score and time taken to find the optimal hyperparameters.

The importance of multi-class prediction cannot be overstated, especially in the context of gesture recognition. Accurately segmenting gestures into phases such as Rest, Preparation, Stroke, Hold, and Retraction is crucial not only for linguistic and computational analysis but also for broader applications such as sign language recognition, human-computer interaction, and behavior recognition. These gesture phases carry important meaning, and small variations in movement can drastically change the intent behind them. Therefore, precise phase segmentation is essential for the success of real-time gesture-based applications.

Given the complexity of the task and the challenges posed by ambiguous phase boundaries and gesture variability across individuals, the use of machine learning models is a significant advancement over manual segmentation methods. By automating the gesture phase segmentation process, this study paves the way for more scalable, consistent, and efficient analyses in various fields, including assistive technologies, human-computer interfaces, and multimodal communication research.