**Instituto Tecnológico**

**y de Estudios Superiores de Occidente**

Official recognition of higher education validity granted by Secretarial Agreement No. 15018, published in the Official Journal of the Federation on November 29, 1976.

Department of Electronics, Systems and Informatics

Master in Computer Systems



**Landmark-driven Transformer Encodings for real-time Mexican Sign Language Recognition.**

WORK SUBMITTED to obtain the DEGREE of

Master in Computer Systems

Presenting: Diego Adrián Santa Cruz Baur

Advisor: Víctor Hugo Martínez Sánchez

Tlaquepaque, Jalisco. July 1st, 2025.

**Instituto Tecnológico**

**y de Estudios Superiores de Occidente**

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Maestría en Sistemas Computacionales



**Codificadores transformer basados en marcadores espaciales para el reconocimiento de la Lengua de Señas Mexicana en tiempo real.**

Trabajo recepcional para obtener el grado de

Maestro en Sistemas Computacionales

Presenta: Diego Adrián Santa Cruz Baur

Asesor: Víctor Hugo Martínez Sánchez

Tlaquepaque, Jalisco. 1 de julio, 2025.

ACKNOWLEDGEMENTS

The author wishes to thank his advisor for his wisdom, kindness, and patience throughout this process, as well as the professors and staff of ITESO for their commitment to the integral development of their students.

The author extends his appreciation to his team at Oracle for the flexibility, encouragement, and backing they provided.  
  
The author is thankful to his sister for her generous attention, which helped him navigate the ideas that shaped this work.

Finally, the author expresses deep gratitude to his parents for their constant support and belief in him.

AGRADECIMIENTOS

El autor desea agradecer a su asesor por su sabiduría, amabilidad y paciencia a lo largo de este proceso, así como a los profesores y al personal del ITESO por su compromiso con el desarrollo integral de sus estudiantes.

El autor extiende su agradecimiento a su equipo en Oracle por la flexibilidad y el respaldo brindados.

El autor agradece a su hermana por su generosa atención, que le ayudó a navegar las ideas que dieron forma a este trabajo.

Finalmente, el autor expresa su profunda gratitud a sus padres por su constante apoyo y la confianza que le brindaron.

DEDICATION

To my parents.

DEDICATORIA

A mis padres.

ABSTRACT

This report presents a system for real-time recognition of isolated Mexican Sign Language (MSL) signs using lightweight transformer encoders. The system focuses on alphanumeric signs—both static and dynamic—and is designed to run on standard consumer hardware.

The project is grounded in a morphological description of MSL signs as compositions of six units, among them hand configuration and motion. Using this framework, we extract hand and body landmarks from video using MediaPipe. These landmarks are optionally transformed to isolate specific components of sign structure, and may then be processed by one of many dimensionality reduction techniques to produce lighter feature representations. The composition of these decisions results in hundreds of dataset variants.

To select the most promising ones, we apply a two-stage pruning process: first using simple K-Nearest Neighbors classifiers and analysis using statistical performance metrics, then evaluating transformer models trained on the strongest candidates. We test five compact encoder architectures—BERT-tiny, mini, small, medium, and DistilBERT. The top-scoring model, BERT-Mini trained on unprocessed coordinates, achieves 97.4% accuracy. However, this does not translate into live-inference conditions, but another model, which attains a lesser 89.0% in testing, shows better robustness to signer variability and overall performance.

We conclude with a discussion on the effects of the geometric transformation process explored in this study, and propose future work to understand and improve them and to develop more robust, performant, general-use models.

RESUMEN

Este reporte presenta un sistema para el reconocimiento en tiempo real de la Lengua de Señas Mexicana (LSM) utilizando codificadores transformer. El trabajo se enfoca en señas alfanuméricas aisladas, tanto estáticas como dinámicas, y está pensado para funcionar en equipos de uso general.

El proyecto se basa en una descripción morfológica de las señas de LSM como composiciones de seis unidades, entre ellas la configuración manual y el movimiento. Bajo este marco teórico, se extraen coordenadas de puntos clave de las manos y el cuerpo utilizando MediaPipe. Estas coordenadas se transforman opcionalmente para aislar componentes específicos de la estructura de las señas, y pueden ser procesadas con una entre muchas técnicas de reducción dimensional para generar representaciones más compactas. La composición de estas decisiones produce cientas de variantes del conjunto de datos.

Para seleccionar las más prometedoras, se aplica un filtrado progresivo en dos etapas: primero con clasificadores K-Nearest Neighbors y analizando métricas estadísticas, y luego entrenando modelos transformer sobre las variantes más destacadas. Se evalúan cinco arquitecturas compactas: BERT-tiny, mini, small, medium y DistilBERT. El modelo con mayor precisión, BERT-mini entrenado sobre coordenadas no procesadas, logra 97.4 % de exactitud en pruebas. Sin embargo, esto no se traduce a la inferencia en tiempo real, pero otro modelo, que en pruebas alcanza una exactitud de solo 89.0%, tiende a ser más robusto ante la variabilidad de la persona señante y muestra mejor rendimiento en general.

Concluimos con una discusión sobre los efectos de las transformaciones geométricas exploradas en este estudio, y proponemos trabajos futuros para comprenderlas y mejorarlas, así como para desarrollar modelos más robustos, eficientes y de aplicación más extensa.

TABLE OF CONTENTS

[ACKNOWLEDGEMENTS 3](#_Toc202210532)

[AGRADECIMIENTOS 4](#_Toc202210533)

[DEDICATION 5](#_Toc202210534)

[DEDICATORIA 6](#_Toc202210535)

[ABSTRACT 7](#_Toc202210536)

[RESUMEN 8](#_Toc202210537)

[TABLE OF CONTENTS 9](#_Toc202210538)

[LIST OF FIGURES 11](#_Toc202210539)

[LIST OF TABLES 14](#_Toc202210540)

[LIST OF ACRONYMS AND ABBREVIATIONS 15](#_Toc202210541)

[1. INTRODUCTION 16](#_Toc202210542)

[1.1. Mexican Sign Language Background 18](#_Toc202210543)

[2. STATE OF THE ART 24](#_Toc202210544)

[2.1. Historical Progression in Sign Language Recognition 25](#_Toc202210545)

[2.2 Mexican Sign Language Recognition 27](#_Toc202210546)

[2.3 Recent Advances in International Sign Language Recognition 29](#_Toc202210547)

[3. THEORETICAL FRAMEWORK 33](#_Toc202210548)

[3.1 Morphological Structure of Mexican Sign Language 34](#_Toc202210549)

[3.2 Feature Extraction and Representation 34](#_Toc202210550)

[3.2.1 Convolution Neural Networks (CNN) 34](#_Toc202210551)

[3.2.2 Landmark Extraction with MediaPipe 34](#_Toc202210552)

[3.3 Dimensionality Reduction of Features 35](#_Toc202210553)

[3.4 Neural Network Architectures for Sequential Modeling 35](#_Toc202210554)

[3.5 Evaluation Metrics for Classification Models 36](#_Toc202210555)

[4. METHODOLOGY 38](#_Toc202210556)

[4.1 Methodology Overview 39](#_Toc202210557)

[4.2 Raw Dataset 40](#_Toc202210558)

[4.2.1 Data Organization and Filtering 41](#_Toc202210559)

[4.3 Data Preparation 42](#_Toc202210560)

[4.3.1 Preparation Overview 42](#_Toc202210561)

[4.3.2 (PH1) Video to Landmarks 42](#_Toc202210562)

[4.3.2.1 Purpose of Using Landmarks instead of Images 42](#_Toc202210563)

[4.3.2.2 Basic Use of MediaPipe 43](#_Toc202210564)

[4.3.2.3 Use of MediaPipe for this Project 44](#_Toc202210565)

[4.3.2.4 PH1 Output Description 46](#_Toc202210566)

[4.3.3 (PH2) Geometric Landmark Transformations 46](#_Toc202210567)

[4.3.3.1 PH2 Output Description 48](#_Toc202210568)

[4.3.4 (PH3) Dimensionality Reduction 48](#_Toc202210569)

[4.3.4.1 3-Component Reduction Visualization Insights 49](#_Toc202210570)

[4.4 Dataset and Model Selection 55](#_Toc202210571)

[4.4.1 Goal 55](#_Toc202210572)

[4.4.2 Dataset Selection 56](#_Toc202210573)

[4.4.2.1 Selection of Subsets Using K-Means 57](#_Toc202210574)

[4.4.2.2 Fitting K-Nearest Neighbors Models for each Subset, for each Dataset Candidate 57](#_Toc202210575)

[4.4.2.2.1 Computing Significance Scores based on Observed Accuracy 58](#_Toc202210576)

[4.4.2.2.2 Comparing Models across Number of Classes 60](#_Toc202210577)

[4.4.2.2.3 Active Hand Detection Model 68](#_Toc202210578)

[4.4.2.2.4 Most accurate KNN Sign Recognition Models 68](#_Toc202210579)

[4.4.3 Model Selection 69](#_Toc202210580)

[4.4.4 Final Selection 72](#_Toc202210581)

[4.5 Selected Model Finetuning 73](#_Toc202210582)

[5. RESULTS AND DISCUSSION 74](#_Toc202210583)

[5.1 Results 75](#_Toc202210584)

[5.2 Discussion 76](#_Toc202210585)

[5.2.1 Comparing to Similar Studies 76](#_Toc202210586)

[5.2.2 Interpretation of the Confusion Matrix 78](#_Toc202210587)

[5.2.3 Live-Inference System Reliability and Limitations 81](#_Toc202210588)

[5.2.4 Comparison between KNN and BERT-mini and Implications on Geometric Transformations 84](#_Toc202210589)

[6. CONCLUSIONS AND FUTURE WORK 86](#_Toc202210590)

[6.1. Conclusions 87](#_Toc202210591)

[6.2. Future work 88](#_Toc202210592)

[7. Bibliography 90](#_Toc202210593)

LIST OF FIGURES

[Figure 1: MSL alphabet, from [15] 19](#_Toc202210594)

[Figure 2: MSL numbers, from [15] 19](#_Toc202210595)

[Figure 3: Possible hand orientations in MSL, as described by Cruz Aldrete et al. [10] 21](#_Toc202210596)

[Figure 4: Hand landmarks extracted by MediaPipe 26](#_Toc202210597)

[Figure 5: Methodology overview flowchart 40](#_Toc202210598)

[Figure 6: Data organization and filtering flowchart 41](#_Toc202210599)

[Figure 7: Data preparation 42](#_Toc202210600)

[Figure 8: MediaPipe hand landmarks 43](#_Toc202210601)

[Figure 9: MediaPipe pose landmarks 43](#_Toc202210602)

[Figure 10: Phase 1: landmark extraction 44](#_Toc202210603)

[Figure 11: Phase 2: geometric landmark transformations 47](#_Toc202210604)

[Figure 12: PH3: dimensionality reduction 49](#_Toc202210605)

[Figure 13: 3-component reduction of PH2-processed per-frame data using PCA with a cosine kernel and coloring by sign 50](#_Toc202210606)

[Figure 14: 3-component reduction of non-PH2-processed per-frame data using PCA with a cosine kernel and coloring by sign 50](#_Toc202210607)

[Figure 15: Using a cosine kernel 51](#_Toc202210608)

[Figure 16: Using a polynomial kernel 51](#_Toc202210609)

[Figure 17: Using an RBF kernel 51](#_Toc202210610)

[Figure 18: Using a sigmoid kernel 51](#_Toc202210611)

[Figure 19: Using a cosine kernel 52](#_Toc202210612)

[Figure 20: Using a polynomial kernel 52](#_Toc202210613)

[Figure 21: Using an RBF kernel 52](#_Toc202210614)

[Figure 22: Using a sigmoid kernel 52](#_Toc202210615)

[Figure 23: 3-component reduction of the PH2-processed per-frame data using PCA with an RBF kernel and coloring by sign 53](#_Toc202210616)

[Figure 24: 3-component reduction of the non-PH2-processed per-frame data using PCA with an RBF kernel and coloring by sign 53](#_Toc202210617)

[Figure 25: 3-component reduction of the PH2-processed per-video data using PCA with an RBF kernel and coloring by sign 54](#_Toc202210618)

[Figure 26: 3-component reduction of the non-PH2-processed per-video data using PCA with an RBF kernel and coloring by sign 54](#_Toc202210619)

[Figure 27: 3-component reduction of the PH2-processed per-frame data using UMAP and coloring by sign 55](#_Toc202210620)

[Figure 28: 3-component reduction of the PH2-processed per-video data using UMAP and coloring by sign 55](#_Toc202210621)

[Figure 29: Dataset and model selection 56](#_Toc202210622)

[Figure 30: Dataset Selection 57](#_Toc202210623)

[Figure 31: Mean of the top 10 significance scores for each number of classes studied 60](#_Toc202210624)

[Figure 32: Evolution over the number of classes of the significance scores obtained for data trained on per-frame (blue) or per-video (orange) data 61](#_Toc202210625)

[Figure 33: Evolution over the number of classes of the significance scores obtained for data passed through PH2 (orange) vs data that did not (blue) 62](#_Toc202210626)

[Figure 34: Evolution over the number of classes of the significance scores obtained for data passed through PH3 (orange) vs data that did not (blue) 63](#_Toc202210627)

[Figure 35: Evolution over the number of classes of the significance scores obtained using different reduction techniques 63](#_Toc202210628)

[Figure 36: Evolution over the number of classes of the significance scores obtained combining different kernels with PCA 65](#_Toc202210629)

[Figure 37: Mean of the top 10 accuracy scores for different number of components in a PCA reducer 66](#_Toc202210630)

[Figure 38: Scree Plot: Variance Explained by PCA Components (non-geometrically transformed data) 67](#_Toc202210631)

[Figure 39: Scree Plot: Variance Explained by PCA Components (geometrically transformed data) 67](#_Toc202210632)

[Figure 40: Mean of the top 3 accuracies by loaded model 70](#_Toc202210633)

[Figure 41: Mean of the top 3 accuracies by unit of data. Blue: per-frame; orange: per-video. 71](#_Toc202210634)

[Figure 42: Top 3 Mean accuracy for each boolean pair of (PH2, PH3) use as part of data preparation, for all model architectures 72](#_Toc202210635)

[Figure 43: Training loss evolution for most accurate model 75](#_Toc202210636)

[Figure 44: Confusion matrix for most accurate model 76](#_Toc202210637)

[Figure 45: MSL alphabet, from [15] 78](#_Toc202210638)

[Figure 46: MSL numbers, from [15] 78](#_Toc202210639)

[Figure 51: No PH2, active hand detection, PCA, no kernel 98](#_Toc202210640)

[Figure 52: No PH2, active hand detection, PCA, cosine kernel 98](#_Toc202210641)

[Figure 53: No PH2, active hand detection, PCA, polynomial kernel 99](#_Toc202210642)

[Figure 54: No PH2, active hand detection, PCA, RBF kernel 99](#_Toc202210643)

[Figure 55: No PH2, active hand detection, PCA, sigmoid kernel 99](#_Toc202210644)

[Figure 56: No PH2, active hand detection, UMAP 99](#_Toc202210645)

[Figure 57: No PH2, per-frame sign recognition, PCA, no kernel 100](#_Toc202210646)

[Figure 58: No PH2, per-frame sign recognition, PCA, cosine kernel 100](#_Toc202210647)

[Figure 59: No PH2, per-frame sign recognition, PCA, polynomial kernel 100](#_Toc202210648)

[Figure 60: No PH2, per-frame sign recognition, PCA, RBF kernel 100](#_Toc202210649)

[Figure 61: No PH2, per-frame sign recognition, PCA, sigmoid kernel 101](#_Toc202210650)

[Figure 62: No PH2, per-frame sign recognition, UMAP 101](#_Toc202210651)

[Figure 63: No PH2, per-video sign recognition, PCA, no kernel 101](#_Toc202210652)

[Figure 64: No PH2, per-video sign recognition, PCA, cosine kernel 101](#_Toc202210653)

[Figure 65: No PH2, per-video sign recognition, PCA, polynomial kernel 102](#_Toc202210654)

[Figure 66: No PH2, per-video sign recognition, PCA, RBF kernel 102](#_Toc202210655)

[Figure 67: No PH2, per-video sign recognition, PCA, sigmoid kernel 102](#_Toc202210656)

[Figure 68: No PH2, per-video sign recognition, UMAP 102](#_Toc202210657)

[Figure 69: PH2 transformed, active hand detection, PCA, no kernel 103](#_Toc202210658)

[Figure 70: PH2 transformed, active hand detection, PCA, cosine kernel 103](#_Toc202210659)

[Figure 71: PH2 transformed, active hand detection, PCA, polynomial kernel 103](#_Toc202210660)

[Figure 72: PH2 transformed, active hand detection, PCA, RBF kernel 103](#_Toc202210661)

[Figure 73: PH2 transformed, active hand detection, PCA, sigmoid kernel 104](#_Toc202210662)

[Figure 74: PH2 transformed, active hand detection, UMAP 104](#_Toc202210663)

[Figure 75: PH2 transformed, per-frame sign recognition, PCA, no kernel 104](#_Toc202210664)

[Figure 76: PH2 transformed, per-frame sign recognition, PCA, cosine kernel 104](#_Toc202210665)

[Figure 77: PH2 transformed, per-frame sign recognition, PCA, polynomial kernel 105](#_Toc202210666)

[Figure 78: PH2 transformed, per-frame sign recognition, PCA, RBF kernel 105](#_Toc202210667)

[Figure 79: PH2 transformed, per-frame sign recognition, PCA, sigmoid kernel 105](#_Toc202210668)

[Figure 80: PH2 transformed, per-frame sign recognition, UMAP 105](#_Toc202210669)

[Figure 81: PH2 transformed, per-video sign recognition, PCA, no kernel 106](#_Toc202210670)

[Figure 82: PH2 transformed, per-video sign recognition, PCA, cosine kernel 106](#_Toc202210671)

[Figure 83: PH2 transformed, per-video sign recognition, PCA, polynomial kernel 106](#_Toc202210672)

[Figure 84: PH2 transformed, per-video sign recognition, PCA, RBF kernel 106](#_Toc202210673)

[Figure 85: PH2 transformed, per-video sign recognition, PCA, sigmoid kernel 107](#_Toc202210674)

[Figure 86: PH2 transformed, per-video sign recognition, UMAP 107](#_Toc202210675)

[Figure 87: Confusion matrix for BERT-mini model trained on per-frame data, with no PH2, with no PH3 108](#_Toc202210676)

[Figure 88: Training loss evolution for BERT-mini model trained on per-frame data, with no PH2, with no PH3 108](#_Toc202210677)

[Figure 89: Confusion matrix for BERT-mini model trained on per-frame data, with no PH2, with PH3 109](#_Toc202210678)

[Figure 90: Training loss evolution for BERT-mini model trained on per-frame data, with no PH2, with PH3 109](#_Toc202210679)

[Figure 91: Confusion matrix for BERT-mini model trained on per-frame data, with PH2, with no PH3 110](#_Toc202210680)

[Figure 92: Training loss evolution for BERT-mini model trained on per-frame data, with PH2, with no PH3 110](#_Toc202210681)

[Figure 93: Confusion matrix for BERT-mini model trained on per-frame data, with PH2, with PH3 111](#_Toc202210682)

[Figure 94: Training loss evolution for BERT-mini model trained on per-frame data, with PH2, with PH3 111](#_Toc202210683)

[Figure 95: Confusion matrix for BERT-mini model trained on per-video data, with no PH2, with no PH3 112](#_Toc202210684)

[Figure 96: Training loss evolution for BERT-mini model trained on per-video data, with no PH2, with no PH3 112](#_Toc202210685)

[Figure 97: Confusion matrix for BERT-mini model trained on per-video data, with no PH2, with PH3 113](#_Toc202210686)

[Figure 98: Training loss evolution for BERT-mini model trained on per-video data, with no PH2, with PH3 113](#_Toc202210687)

[Figure 99: Confusion matrix for BERT-mini model trained on per-video data, with PH2, with no PH3 114](#_Toc202210688)

[Figure 100: Training loss evolution for BERT-mini model trained on per-video data, with PH2, with no PH3 114](#_Toc202210689)

[Figure 101: Confusion matrix for BERT-mini model trained on per-video data, with PH2, with PH3 115](#_Toc202210690)

[Figure 102: Training loss evolution for BERT-mini model trained on per-video data, with PH2, with PH3 115](#_Toc202210691)

[Figure 103: Confusion matrix for KNN model trained on per-frame data, with no PH2, with no PH3 116](#_Toc202210692)

LIST OF TABLES

[Table 1: Description of alphanumeric signs by element 22](#_Toc202211757)

[Table 2: State of the Art MSL recognition systems comparison 28](#_Toc202211758)

[Table 3: State of the Art Sign Language Recognition systems comparison 31](#_Toc202211759)

[Table 4: Samples from the alphanumeric MSL dataset [46], [47] 39](#_Toc202211760)

[Table 5: Comparison of PH2-processed per-frame data using PCA and 4 different kernels 50](#_Toc202211761)

[Table 6: Comparison of non-PH2-processed per-frame data using PCA and 4 different kernels 51](#_Toc202211762)

[Table 7: Size comparison of different BERT inspired architectures (see [59]) 55](#_Toc202211763)

[Table 8: Active hand detection models best scores 67](#_Toc202211764)

[Table 9: Scores for best KNN alphanumeric sign classifiers 67](#_Toc202211765)

[Table 10: Final dataset selection scoring metrics 71](#_Toc202211766)

[Table 11: Metrics for most accurate model in testing 74](#_Toc202211767)

[Table 12: Comparison with Rodriguez et al. [45] study 75](#_Toc202211768)

LIST OF ACRONYMS AND ABBREVIATIONS

|  |  |  |
| --- | --- | --- |
| ASL |  | American Sign Language |
| BERT |  | Bidirectional Encoder Representations from Transformers |
| BLEU |  | Bilingual Evaluation Understudy |
| CPU |  | Central Processing Unit |
| CSL |  | Chinese Sign Language |
| CNN |  | Convolutional Neural Network |
| DGS |  | German Sign Language, by its german acronym |
| DNN |  | Deep Neural Network |
| DTW |  | Dynamic Time Warping |
| GBL |  | Gaussian Bayesian Linear |
| GRU |  | Gated Recurrent Unit |
| HMM |  | Hidden Markov Models |
| LSTM |  | Long Short-Term Memory |
| ML |  | Machine Learning |
| MSL |  | Mexican Sign Language |
| PH1 |  | Phase 1 |
| PH2 |  | Phase 2 |
| PH3 |  | Phase 3 |
| PCA |  | Principal Component Analysis |
| RBF |  | Radial-Basis Function |
| RNN |  | Recurrent Neural Network |
| RGB |  | Red-Green-Blue |
| UMAP |  | Uniform Manifold Approximation and Projection |
| ViT |  | Vision Transformer |
| WER |  | Word Error Rate |
| WLASL |  | Word-Level American Sign Language |

# INTRODUCTION

Sign languages are full-fledged natural languages that develop organically within Deaf communities worldwide. They use visual signals—primarily articulated through hand gestures—to convey meaning rather than relying on spoken sounds [1]. MSL is the native sign language of Mexico’s Deaf population [2]. While approximately 2.3 million people in Mexico were reported to have some form of hearing disability in 2021 [3], it is estimated that around 300,000—deaf or otherwise—use MSL as a means of communication, an indication of the language’s reach relative to both the hard-of-hearing population (13 %) and the general population (0.2 %) [3], [4]. Enabling communication between Deaf and hearing people is key to fostering inclusion in education, professional settings, and public life at large.

Creating real-time sign-language recognition systems can provide the foundation for simple educational and translation tools, improving access to classrooms, workplaces, and public services, and is an active area of research worldwide [5], [6].

Ultimately, the ambition for these systems is to tackle the full complexities of sign language. However, collecting and annotating the large, richly contextualized datasets required for these capabilities remains a significant challenge [5], [6], [7]. For example, some of the more advanced components of sign languages, such as their grammar, require continous sign language capture—where each sample corresponds to full sentences or beyond. While only a handful of such datasets exist globally [8], MSL has seen one notable effort: González-Rodríguez et al. [9] assembled 1 000 video samples for ten simple sentences using synchronized depth cameras; however, this resource is not publicly available. Other commonplace aspects of sign language like pointing signs [10]—where the signer will point directly to a specific object, person or place to refer to it—are also challenging to account for and, as far as the authors of this work are aware of, have not been addressed by any sign language recognition system.

As a result, many studies focus on more tractable subproblems for which annotated data exists. Particularly relevant to this work is the prior MSL research which has tackled the following challenges:

* Isolated Sign Language Recognition
  + Publicly accessible corpora in MSL concentrates on Isolated Sign Language – where each sample contains a single sign. Therefore most research in the field has concentrated on this subproblem [11], [7], [12], [13].
* Dynamic Sign Language Recognition
  + A division can be made between signs that require movement (called “dynamic”) and those that do not (called “static”). The former adds a complexity to the challenge that many modern Machine Learning (ML) architectures are well equipped to address, and therefore most recent works in the area focus on both dynamic and static signs, whether with separate models, as Rodriguez et al. did [11], or jointly, as Martínez-Sánchez [7] et al. and Mejía-Pérez et al. [12] did.

In this work, we concentrate on isolated sign recognition of both dynamic and static alphanumeric signs with a single model.

Drawing on the distinctive features of MSL outlined by Cruz Aldrete et al. [10], our approach treats each sign at the morphological level as a composition of six minor units:

* Hand Configuration
* Place of Articulation
* Hand Motion Type
* Direction of Hand Motion
* Hand Orientation
* Non-Manual Cues

We apply a complex data preparation methodology to extract the features units that concern our videos, and use it to develop this project’s main contribution: an accurate, real-time recognition unified system for both static and dynamic alphanumeric MSL signs that runs on consumer-grade hardware. Throughout our development, we also conduct experiments to evaluate the hypothesis that preprocessing focused on distilling the six minor units of MSL enhances model performance.

The remainder of this report is organized as follows. Chapter 2 reviews the evolution of sign language recognition systems and summarizes prior work on MSL. Chapter 3 presents the theoretical framework, covering the linguistic structure of signs, feature extraction methods, dimensionality reduction techniques, modeling approaches, and evaluation metrics. Chapter 4 details the methodology used for dataset construction, preprocessing, and model training. Chapter 5 reports and analyzes the experimental results, discussing the project’s limitations. Finally, Chapter 6 offers conclusions and outlines potential directions for future work.

## Mexican Sign Language Background

MSL is the predominant language of the Deaf community in Mexico [10]. It originated with the foundation of the Escuela Nacional de Sordomudos in Mexico City in 1867 [10], and is a fully developed natural language with its own grammar and lexicon, distinct from spoken Spanish [10]. MSL was officially recognized as a national language by the mexican state in 2005 [14], and it exhibits regional variation across communities—most notably along the rural–urban divide [10].

This research focuses on a specific subset of MSL lexicon: alphanumeric signs. It includes the 26 signs corresponding to the letters in the English alphabet, as well as “LL” and “RR”. Although “Ñ” is generally considered to be part of the MSL alphabet and has its own sign, it is not present in the dataset we used. Additionally, our dataset also includes the signs for the numbers from 0 to 10, although we will not use the data for 0 and 7 for reasons discussed later. Visual representations of these signs, made by IncluSor [15], can be seen in Figure 1 and Figure 2.

|  |  |
| --- | --- |
| Figure 1: MSL alphabet, from [15] | Figure 2: MSL numbers, from [15] |

To understand the impact of choosing these alphanumeric signs for the scope of our project, this section discusses how signs are generally categorized and the portions that the subset we chose covers.

Cruz Aldrete et al. [10] propose comprehensive guidelines to describe signs as combination of the following features:

* Hand Configuration
* Place of Articulation
* Hand Motion Type
* Direction of Hand Motion
* Hand Orientation
* Non-Manual Cues

The first five features can be applied to each of the two hands, and combining this either symmetrically or freely adds further complexities [10]. In the case of our project, we have limited ourselves to single-hand sign recognition. Although a path to extend the system to bimanual sign recognition is easy to envision using the same tools we build our present system on, it would require a complete reconfiguration of our pipeline. This is the main limitation in terms of potential for rapidly extending the application of our project, but it also served to clearly define the boundaries for the development workflow.

For each of the features, Cruz Aldrete et al. [10] provide what they claim to be a complete listing of the options in use for MSL. For the Hand Configuration element, Cruz Aldrete et al. identify 30 groupings with each one having between 3 and 26 variants. For Hand Orientation, 9 options are shown which can be seen in Figure 3. For the Place of Articulation, 39 points are specified, primarily in the face, torso, arms and hands areas. The Direction of Motion element refers to the trajectory the hand follows throughout the articulation of the sign, and 10 options are identified, not accounting for signs in which the hand does not move. Then, 23 different movement patterns relating to the fingers, hands, wrists and arms are described, not accounting for static signs. Finally, 13 groups of Non-Manual Cue actions are grouped, mostly based on the body involved in the action (eyes, nose, eyebrows, …), with each group having from 1 to 12 specific actions.

Achieving substantial coverage on all of these components is unrealistic given the current publicly available MSL datasets. This research focuses on hand configuration for three main reasons. First, when developing a complete sign language recognition system under the framework that decomposes signs into the 6 previously mentioned units, it is apparent that hand configuration is essential because, while not every hand includes movement for example, it will always have a meaningful hand shape. In particular, whereas static “movement”, the area in front of the chest, and a “neutral” facial expression and pose can be considered to be the “default” for their respective units, there is no “default” hand configuration or hand orientation. Thus, it is a necessity for any sign language system based on this theoretical framework to effectively model hand configuration. Second, the human hand—with its several fingers that move both independently and together—poses a challenging pattern-recognition problem which leads to new understanding of sign language as well as ML concepts. Third, modern real-time hand-landmarking models produce vector outputs that map directly to handshape descriptors, opening up opportunities to investigate this approach.

Amongst the signs in our dataset, we observe variance for 4 of the 6 features: Hand Configuration, Hand Motion, Direction of Motion and Hand Orientation. A case by case for each of the signs can be seen in Table 1.

Out of the 30 hand configuration groupings listed by Cruz Aldrete et al. [10], our dataset covers 25 of them, missing only the “I-L”, “Dedo Medio”, “Dedo Anular”, “I-L” and “Pico” configurations. As mentioned, this was the driving factor in the choice of our dataset. Additionally, although alphanumeric signs are not particularly prevalent in common MSL communication [10], their hand configurations do carry a special importance in that some signs use the hand configuration for the first letter of the word in Spanish corresponding to the sign in a phenomenon called “initialized signs” [10]. Although by no means conclusive data, in a comparison between American Sign Language (ASL) and MSL, Martínez-Sánchez et al. noted that out of 100 words, 12 were initialized in ASL while 37 were initialized for MSL [7].

For hand orientation, 6 out of the 9 options were covered, with a heavy presence of signs where the palm faces the camera, with the fingers pointing up (orientation number 7 in Figure 3).

|  |
| --- |
| Figure 3: Possible hand orientations in MSL, as described by Cruz Aldrete et al. [10] |



For hand placement, there was minimal coverage, since for all of the alphanumeric signs selected the area of articulation is not of particular importance and generally defaults to the area in front of the chest. For the two “double letter” signs (“LL” and “RR”), there is somewhat of a specificity in terms of area however, in that they both use the “double letter” motion in which the hand moves from one side of the chest to the other.

Similarly, there is very little variance in terms of direction of movement. All of the signs, even those which include movement, happen in a single place, except for the “X” sign, which involves a “pulling” like movement with the direction pointing backwards in the depth axis.

We observe slightly better variation in terms of hand motion: only 5 of the 24 movement types appear in our dataset, and most signs remain static. Although our focus is on hand configuration, motion recognition is a very interesting challenge: modeling the temporal dynamics and subtle trajectories of signs demands sophisticated techniques, and extending a recognition system to continuous signing adds the extra obstacle of identifying the beginning and end of signs, which relies on movement pattern identification for accurate segmentation [16].

Finally, although alphanumeric signs don’t contain any mandatory non-manual cues, the signers in our dataset will often mouth the letter in Spanish corresponding to the alphabetical signs, an action which is not uncommon for this particular set of signs given their heavy connection to Spanish [10].

Table 1: Description of alphanumeric signs by element

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sign | Hand configuration | Hand orientation | Movement description | Movement direction |
| A | A | 7 | Static | In place |
| B | B | 7 | Static | In place |
| C | C | 5 | Static | In place |
| D | D | 5 | Static | In place |
| E | E | 7 | Static | In place |
| F | F | 7 | Static | In place |
| G | L | 9 | Static | In place |
| H | H | 9 | Static | In place |
| I | I | 7 | Static | In place |
| J | I | 7->6 | Wrist rotation | In place |
| K | P | 5<->6 | Repeated “wrist nodding” | In place |
| L | L | 7 | Static | In place |
| LL | L | 7 | Linear | Side-to-side |
| M | M | 4 | Static | In place |
| N | N | 4 | Static | In place |
| O | O | 5 | Static | In place |
| P | P | 5 | Static | In place |
| Q | Q | 4<->7 | Repeated “wrist nodding” | In place |
| R | R | 7 | Static | In place |
| RR | R | 7 | Linear | Side-to-side |
| S | S | 7 | Static | In place |
| T | T | 7 | Static | In place |
| U | U | 7 | Static | In place |
| V | 2 | 7 | Static | In place |
| W | 3 | 7 | Static | In place |
| X | Q | 6 | Linear | Pull towards the body (depth axis) |
| Y | Y | 8 | Static | In place |
| Z | 1 | 7 | Zig-zag | Zig-zag |
| 0 | O | 5 | Static | In place |
| 1 | 1 | 8 | Static | In place |
| 2 | 2 | 8 | Static | In place |
| 3 | 3 | 8 | Static | In place |
| 4 | 4 | 8 | Static | In place |
| 5 | 5 | 8 | Static | In place |
| 6 | A | 9 | Static | In place |
| 7 | H | 9 | Static | In place |
| 8 | 8 | 9 | Static | In place |
| 9 | B->S | 8 | Retraction of fingers | In place |
| 10 | 5 | 8->4 | Wrist rotation | Down |

# STATE OF THE ART

## Historical Progression in Sign Language Recognition

The evolution of sign language recognition systems has closely followed advancements in computer science and ML.

Early research in the field often approached the problem using statistical models. For example, in 1995 Starner et al. [17] achieved real-time recognition of ASL word signs using Hidden Markov Models (HMM). This model is intended to be used on sequential data where it can be assumed that the relationship between successive observations contains valuable information [18]. It has been widely used for phenomena for which observations can be made over time, making it specially well suited for isolated dynamic signs [6]. However, HMMs have been shown to scale poorly to large amounts of data because it becomes computationally costly to train these models and use them for inference [18]. They also do not perform well at identifying long-range relationships between observations, making it less effective for continuous sign language recognition [18]. It is also important to note that at this point, the data on which these models were trained on was not directly the images or videos as is often the case today, but rather handcrafted features laboriously derived from the data with the use of special equipment and requiring domain-specific knowledge [19].

In the early 2000s, researchers began exploring alternative ML models such as Support Vector Machines (SVMs) [20] and K-Nearest Neighbors (KNN) [6]. These models offered lower computational cost and a higher potential for scalability as compared to HMMs but lost the intrinsic sequential component that helped HMMs in dealing with dynamic signs. Therefore, advances were mainly made in static sign language recognition. These models also still relied on the manual extraction of hand-crafted features. In the case of KNN, this is because it’s mode of operation is to compare instances in the data and find the most similar instances based on some distance metric, but it has no understanding of the structures within the data. For SVMs, it tries to find a way to separate classes as much as possible based on the raw data, but, again, it has no understanding of the structures within the data.

The popularization of neural networks in ML in the late 2000s and early 2010s was a major breakthrough in sign language recognition because it significantly reduced the need to manually extract hand-crafted features [6]. Neural networks began to use raw pixel information from images (or video frames) as direct input, allowing them to learn patterns directly from the data, rather than relying on manual feature extraction [21]. Unlike traditional ML models, neural networks are especially well-suited to identify the structures and patterns within data. Their architecture, particularly in the case of Deep Neural Networks (DNN) [21], enables them to learn features of increasing complexity layer by layer. This shift opened the field to a broader range of researchers, enabling them to move beyond the labor-intensive process of data preparation and focus on exploring the unique challenges of sign language recognition that could be better addressed by more sophisticated neural network architectures.

Convolutional Neural Networks (CNNs) [22], [23] became especially prominent in sign language recognition in the early 2010s, thanks to their ability to exploit spatial structures in images [24]. The core architectural feature of CNNs is the convolutional layer, where small filters are applied to local regions of the image to detect basic patterns like edges, textures and corners [21]. By moving the area this filter acts on across the image like a sliding window, the whole input is covered while the spatial relationship between nearby pixels is preserved. This is particularly useful in the case of sign language recognition, where local patterns in small regions of the image, notably those where hands appear, are highly informative. Another important attribute for CNNs is the fact that the same filters are used on different parts of this image. These filters can be defined by a relatively low number of weights, making the model computationally lighter and usable for real-time inference [21].

|  |
| --- |
| Figure 4: Hand landmarks extracted by MediaPipe |

Sign language recognition projects using CNNs directly to predict classes have obtained excellent results and have repeatedly advanced accuracy standards in the field [5] [25]. But in fact, CNNs have proven highly effective for object detection and classification in images in general, with sign language recognition being only one of the many applications where their ability to exploit spatial patterns has revolutionized ML [21]. Because of this, some projects make use of this architecture for intermediate tasks such as image segmentation to identify areas in the image where hands appear, or even to extract features from the images, and then to use those features as the input for sing language specific models. One end-to-end example was developed by Koller et al. [24], where first a CNN was used to extract features which are then fed into an HMM. Some projects also make use of pre-trained models based on CNNs to extract features. For example Mocialov et al. [16] used the OpenPose library [26] to extract features and then feed them into a Long Short-Term Memory (LSTM) model. Other studies, like Rodriguez et al. [11], make use of the Google MediaPipe framework which provides pre-trained CNN models for problems such as hand landmarking, pose detection, and face landmarking [27], [28]. The hand landmarking model extracts three dimensional coordinates for the 21 points which can be seen in Figure 4 from images containing hands. This eliminates the need for the labor-intensive process of manually extracting handcrafted features from raw pixel data.

Recurrent Neural Networks (RNN) [29] complement CNNs by exploiting the temporal aspect of video data, making them especially useful for dynamic and continuous sign language recognition. While basic RNNs struggle with capturing long-term dependencies because of vanishing gradients, variants such as LSTM [30] and Gated Recurrent Units (GRU) [31], introduced in the later 2000s and early 2010s, were developed to better handle information over longer sequences [18], [19]. Early systems often combined CNNs to extract spatial features with LSTM or GRU layers to capture temporal patterns, showing improved accuracy in various sign language recognition tasks [20], [21]. However, since RNNs process data sequentially they are difficult to parallelize, slowing down training and inference, which lead researchers to explore Transformer architectures.

Transformers, introduced in 2017 by Vaswani et al. [32], represented a significant shift in how sequential data is processed. Unlike RNNs, which process data sequentially, Transformers use a self-attention mechanism, which allows the model to look at the entire sequence at once. In self-attention, every element in the sequence (for example, each word in a text or each frame in a video) is compared to every other element, regardless of their position. This means the model can decide which parts of the sequence are most relevant to each element, even if they’re far apart. For example, in a sentence like “The cat sat on the mat,” the word “sat” relies on "cat" to understand what is being described. Self-attention allows the model to weigh the importance of the word "cat" when processing "sat", even though they’re not next to each other in the sentence. This ability to directly relate distant parts of the sequence, no matter the range, is what gives Transformers a significant advantage over RNNs, which process the sequence step-by-step and struggle to maintain long-range dependencies. This is especially valuable for tasks like continuous sign language recognition, where signs depend on the overall context of the sequence, as seen in systems like those in the systems of Mendoza et al. [33], Camgoz et al. [34], and Zhang et al. [35]. Despite their advantages, Transformers are computationally demanding, especially for long video sequences, which limits their use in real-time systems. To address this, lightweight variants of BERT (Bidirectional Encoder Representations from Transformers) [36], [37], and DistilBERT—a smaller, faster version distilled from BERT [38]–have been developed to reduce size and inference time while retaining much of the original model's performance.

## Mexican Sign Language Recognition

Research on MSL recognition spans a range of methodologies, data sources, and system goals. In this section, we focus on recent work that is technically relevant to the present study — particularly in terms of input modality, model design, and recognition task. Since a broader historical overview of sign language recognition has already been provided in the previous section, our aim here is to clarify how the current project builds upon or diverges from this recent body of work specific to MSL. Table 2 shows an overview of the different approaches described in this section.

Marquez et al. (2025) [39] developed a mobile application with an interactive learning module that achieved approximately 92% accuracy. Their approach is based on a CNN architecture inspired by the VGGNet [40] CNN architecture and demonstrates the potential of deploying deep learning for MSL recognition on mobile devices without relying on landmark extraction frameworks. These contributions show that mobile-ready solutions for MSL are technically feasible and increasingly accessible.

Several high-accuracy systems have used specialized hardware such as Kinect or depth-sensing cameras to capture rich input data. For example, García-Bautista et al. [13] used Kinect and a Dynamic Time Warping (DTW) [41] algorithm to classify dynamic signs, representing early structured MSL recognition, and achieving an accuracy of 98.6%. Mejía-Pérez et al. [12] later applied RNNs [29] to depth camera data for digit classification, achieving 97.0% accuracy on clean input and 90.3% under noise, marking a shift toward neural modeling with a focus on robustness. More recently, González-Rodríguez et al. [9] combined Kinect video with MediaPipe-based landmark extraction [27] and trained multiple deep learning models, including Transformers [32], to build a continuous MSL translation system that achieved 98.8% accuracy. These systems often incorporate multimodal inputs such as hand, face, and pose landmarks. However, their reliance on specialized sensors limits their accessibility and makes real-world deployment mo re difficult.

To address this, some systems have adopted MediaPipe landmarks to build pipelines that rely entirely on consumer-grade hardware, while maintaining high performance. Cataño et al. [42] achieved between 97.3% and 99.4% accuracy on alphabet recognition using MediaPipe hand landmarks combined with SVM [20] and Random Forest [43] classifiers. However, because the system only uses one frame per sign (the last frame of articulation for dynamic signs), they may lack the temporal context needed for broader recognition tasks involving a wider vocabulary with more complex or movement-based input.

Several of the previously mentioned systems involve creating their own datasets. However, some efforts are more directly focused on developing high-quality data resources for MSL. Martínez-Sánchez et al. [7] developed a dataset covering 100 of the most frequently used MSL signs, totaling 5,000 videos. Their best model, a CNN [22] combined with a Bidirectional LSTM [44], achieved over 99% accuracy in isolated sign classification. This work highlights an emerging interest in vocabularies beyond fingerspelling, previewing a possible reach of MSL recognition systems—whether for translation, education or other purposes—in everyday situations. Additionally, it displays a greater variety of hand movements than other existing MSL datasets, offering a chance to explore this aspect of signing in depth. Nevertheless, as we mentioned in the introduction of this report, our work focuses on the study of hand configuration, and for this particular issue, alphanumeric signs offer excellent coverage of the variety existing in MSL.

The present project builds directly on a line of work initiated by Rodríguez et al. [11], [45], who created and published a dataset of MSL alphabet and number signs [46], [47] and applied MediaPipe-based landmark extraction to train separate models for static and dynamic signs. Their system used a range of classifiers including Gaussian Bayesian Linear model (GBL) [48], SVM [20], LSTM [30], and GRU [31], achieving 94.1% accuracy on static signs and 84.7% on dynamic signs. While the data source and initial preprocessing step are shared, our project extends this work in several directions: we implement a unified model for both static and dynamic signs, explore Transformer [32] architectures, and evaluate alternative data-preprocessing and augmentation strategies.

Table 2: State of the Art MSL recognition systems comparison

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Study | Lexicon | Task Type | Input modality | Methodology | Best accuracy |
| García-Bautista et al. (2017) [13] | 20 words | Isolated | Kinect with depth | DTW | 98.6% |
| Mejía-Pérez et al. (2022) [12] | Digits (0-9) | Isolated | Depth and RGB | RNN | 97.0% clean, 90.3% noisy |
| González-Rodríguez et al. (2024) [9] | 10 multi-word phrases | Continous | Kinect and RGB | MediaPipe + RNN, LSTM, Transformer | 98.8% |
| Marquez et al. (2025) [39] | Alphabet | Isolated | RGB | CNN | 92% |
| Cataño et al. (2024) [42] | Alphabet | Isolated | RGB | MediaPipe + SVM, Random Forest | 99.4% |
| Martínez-Sánchez et al. (2023) [7] | 100 words | Isolated | RGB | CNN + BiLSTM | 99% |
| Rodríguez et al. (2025) [11] | Alphanumeric | Isolated | RGB | MediaPipe + SVM, GRU | 94.1% static, 84.7% dynamic |
| Present work | Alphanumeric | Isolated | RGB | MediaPipe + Transformer | 97.4% |

## Recent Advances in International Sign Language Recognition

The previous section focused on recent research in MSL recognition. To place this work within the broader field, this section reviews international systems in sign language recognition published between in the last 2 years. These works differ in their goals (continuous vs. isolated recognition), model architectures, input types, and the sign languages they target. To make the comparisons clearer, we begin by introducing a set of terms and datasets that will help frame the different approaches.

**Context and Key Concepts**

* Labeling terminology
  + **Gloss:** A written label for a sign; typically a capitalized word from a spoken language like “HOUSE” or “WALK”
* Datasets
  + **CSL-Daily**: A dataset of short phrases in Chinese Sign Language (CSL) used for both isolated and continuous recognition. It contains over 20,000 video samples covering approximately 500 distinct signs.
  + **PHOENIX-Weather 2014/2014T**: A dataset of German Sign Language (DGS, by its german acronym) based on televised wather forecasts annotated with glosses and, in the “T” version, spoken-language translations.
  + **WLASL** (Word-Level American Sign Language): An isolated ASL recognition dataset including over 21,000 video samples and more than 2,000 distinct signs performed by more than 100 signers.
    - WLASL1000 is a subset of WLASL containing 1000 signs, and WLASL100 is a smaller subset of WLASL containing 100 signs
* Input modalities
  + **RGB** (Red-Green-Blue): The raw 3-channel video signal
  + **Landmarks**: coordinate points for the hands, body and/or face extracted from the raw video data
  + **Optical Flow**: Representation of pixel-wise motion between consecutive frames computed from the raw video data
* Evaluation metrics
  + **BLEU** (Bilingual Evaluation Understudy): A metric for evaluating the quality of machine translated text base on n-gram overlap with reference sentences
  + **WER** (Word Error Rate): A metric that compares a predicted sequence of words to a reference by counting insertions, deletions and substitutions

**Continuous Sign Language Recognition**

Camgoz et al. [34] use a transformer-based model that learns to connect sign videos and their meanings by placing them in the same embedding space. Instead of predicting glosses directly, the model learns a shared representation for signs and words, guided by how semantically similar they are. This approach supports recognition and can also help the model roughly identify where signs occur in the video, without needing frame-level annotations.

Zhang et al. [35] extend the transformer model by including a gloss language model as an extra input stream. This helps the system learn not just individual glosses, but also how glosses typically appear together in sequences. Their custom loss function avoids the usual assumption that glosses are independent, making predictions more linguistically coherent.

Jiao et al. [49] introduce CoSign, a lightweight two-stream graph-based model that operates on hand, face and body landmarks. The model uses separate sub-networks for static and dynamic information and includes a regularization mechanism to encourage diverse region-specific features across hand, face, and body landmarks.

**Isolated Sign Language Recognition**

Zuo et al. [50] developed a hybrid CNN and transformer model for isolated sign classification using RGB video, landmark data, and gloss information. The model incorporates semantic label smoothing during training, which softens the target labels based on how semantically similar different glosses are. For example, if the correct gloss is “HOUSE”, a small weight might also be assigned to “HOME”, which is semantically close. This encourages the model to treat confusable signs as partially related, reducing overconfidence and improving generalization. It achieves 93.5% top-1 accuracy on WLASL1000.

Shi et al. [51] use a Vision Transformer (ViT) architecture, which applies self-attention to spatial patches of video frames instead of using convolutional filters. Their model processes RGB, optical flow, and landmark data in parallel, allowing it to capture appearance, motion, and structure simultaneously. This multimodal fusion improves the model’s ability to distinguish signs with subtle visual and movement differences. It is evaluated on WLASL100 and achieves over 93% top-1 accuracy.

Zhou et al. [52] introduce a hierarchical ViT that processes RGB video in stages of increasing abstraction. Instead of analyzing all patches at the same scale, the model gradually reduces spatial resolution while increasing the semantic richness of features, similarly to what CNNs do. This allows it to capture both fine details (like handshape) and broader movements (like arm trajectory) across multiple levels. Trained on CSL-Daily, the model achieves 98.1% top-1 accuracy.

**Comparison of Systems**

Table 3 summarizes these systems, as well as the one in the present work.

Table 3: State of the Art Sign Language Recognition systems comparison

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Study | Language(s) | Task Type | Dataset(s) | Input Modality | Architecture | Best Reported Metric |
| Camgoz et al. (2023) [34] | DGS | Continous | PHOENIX-Weather-2014T | RGB | Transformer | BLEU = 20.8 |
| Zhang et al. (2023) [35] | CSL, DGS | Continuous | CSL-Daily, PHOENIX-Weather-2014 | RGB + semantic gloss embeddings | Transformer | WER = 15.5% (for CSL) |
| Jiao et al. (2023) [49] | CSL, DGS | Continuous | CSL-Daily, PHOENIX-Weather-2014 | Landmarks only | Two-stream Graph Convolutional Network | 88% accuracy (for DGS) |
| Zuo et al. (2023) [50] | ASL, CSL | Isolated | WLASL1000 | RGB + landmarks + softened gloss | CNN + Transformer | 93.5% accuracy |
| Shi et al. (2023) [51] | ASL | Isolated | WLASL100 | RGB + landmarks + optical flow | ViT | 93% accuracy |
| Zhou et al. (2023) [52] | CSL | Isolated | CSL-Daily | RGB | Hierarchical ViT | 98.1% accuracy |
| Present work | MSL | Isolated | MSL Dadictology [46], [47] | Landmarks | Transformer | 97.4% accuracy |

# THEORETICAL FRAMEWORK

In this section, we introduce the theoretical background that supports the decisions made in this work. We organize the section in five parts: the morphological structure of signs in MSL, feature extraction and representation, feature reduction, neural network architectures used in sign language recognition, and the evaluation metrics used to compare models.

## Morphological Structure of Mexican Sign Language

Sign languages are natural human languages that use the body—primarily the hands—to convey meaning instead of sound. Like spoken languages, they have their own grammatical and syntactic rules. In the case of MSL, Cruz Aldrete et al. [10] propose that each sign can be described using six components:

* Hand Configuration
* Place of Articulation
* Hand Motion Type
* Direction of Hand Motion
* Hand Orientation
* Non-Manual Cues

This framework allows for a structured description of signs in a way that supports both linguistic analysis and computational modeling. Each component contributes to meaning, and the combination of features determines the identity of a sign. The components vary in relevance depending on the sign. For example, some signs may include motion or facial expressions, while others may not.

## Feature Extraction and Representation

## Convolution Neural Networks (CNN)

CNNs [22] are a type of neural network designed to efficiently process data with a grid-like structure, such as images. Their key innovation lies in the use of convolutional layers, which apply small, learnable filters across local regions of the input. This approach exploits spatial locality while dramatically reducing the number of parameters through weight sharing. As a result, CNNs can build hierarchical representations of patterns: early layers typically detect edges and simple textures, while deeper layers capture increasingly abstract features like shapes or objects. This structure also confers a degree of translation invariance, allowing the network to recognize patterns regardless of their exact position in the input. Pooling layers, which reduce the spatial dimensions of intermediate representations, enhance robustness to small translations or distortions. For a more detailed theoretical explanation of CNNs and their design principles, see [21], [23] and [53].

## Landmark Extraction with MediaPipe

Instead of working directly with image pixels, this project uses landmark coordinates as input. MediaPipe is a framework developed by Google that uses CNNs to detect landmarks of the hand and body in real-time from regular video input [25], [26]. It returns 21 three-dimensional points per hand and a set of 32 three-dimensional body points. This type of input is lighter than raw images, less sensitive to background noise, and already structured in a way that makes it easier to extract features like hand configuration or hand motion.

## Dimensionality Reduction of Features

High-dimensional inputs such as body or hand landmarks often include noise or irrelevant variation, which can negatively impact model performance and increase computational costs. Dimensionality reduction techniques aim to retain the most informative aspects of the data while discarding redundant or less useful components.

A widely used linear approach is Principal Component Analysis (PCA) [54]. PCA computes the directions, or principal components, along which the data varies the most, and projects the input onto a lower-dimensional subspace spanned by the leading components. This reduces dimensionality while preserving variance and orthogonality, making PCA both efficient and interpretable.

For more complex data structures where linear assumptions fall short, Uniform Manifold Approximation and Projection (UMAP) [55] offers a powerful alternative. UMAP constructs a weighted graph of local relationships and optimizes a low-dimensional embedding that reflects both local topology and global structure. Unlike PCA, UMAP can capture curved or folded manifolds and is particularly well-suited for visualizing high-dimensional clusters or patterns. While slower to compute, its representations often yield better separability for downstream tasks.

## Neural Network Architectures for Sequential Modeling

RNNs are designed to handle sequential data, where the order of inputs matters [29]. Unlike regular neural networks, RNNs maintain a hidden state that is updated over time as new inputs are processed. This allows the network to remember previous information, which is particularly useful for tasks like language modeling and sign language recognition. In the context of the latter, RNNs are useful for recognizing dynamic signs, where hand positions evolve over time across a sequence of frames.

However, regular RNNs face two main problems when learning from long sequences: the vanishing gradient problem and the exploding gradient problem [30]. In the vanishing gradient problem, gradients become extremely small during backpropagation, which makes it hard for the model to learn long-term dependencies. On the other hand, the exploding gradient problem occurs when gradients become too large, leading to unstable model weights and causing the model to diverge. These issues limit the effectiveness of basic RNNs to learn long-range dependencies [30].

LSTM [30] and  GRU [56] were introduced to overcome these issues in standard RNNs. Both LSTM and GRU use gating mechanisms to control how much information is passed along and how much is discarded, allowing them to remember important details over longer sequences. LSTMs are more complex, using three gates: the input gate, the forget gate, and the output gate. This structure helps LSTMs overcome the vanishing and exploding gradient problems by controlling the flow of information more effectively. GRUs are simpler and combine the forget and input gates into a single update gate, but are still highly effective. Both models are widely used for SLR (see [45], [9], [7]) because they can capture temporal dependencies in dynamic signs, which is crucial for accurate recognition.

Transformers [32] provide an alternative to RNNs for sequential modeling. Rather than processing one frame at a time, Transformers use self-attention, which allows the model to look at the entire sequence at once. Self-attention computes the relationships between all parts of the sequence and assigns different weights to each part based on its relevance, enabling the model to focus on the most important information. To compensate for the lack of inherent order in parallel processing, Transformers include positional encodings—which inject information about the position of each element within the sequence into the input. This design allows Transformers to model long-range dependencies more effectively than RNNs while also enabling faster training and inference through parallelism.

BERT (Bidirectional Encoder Representations from Transformers) [36] is a specific implementation of the Transformer model. It is bidirectional, meaning it looks at both the preceding and succeeding context in a sequence, rather than just one direction. This bidirectionality improves the model's understanding of context. BERT uses multiple layers of Transformer encoders, each layer applying self-attention and feedforward networks. In this project, we use BERT-based models to process sequences of landmark coordinates extracted from video frames and classify them into signs. BERT's ability to model complex patterns in sequential data makes it well-suited for sign language recognition tasks [36].

## Evaluation Metrics for Classification Models

Evaluating classification models requires metrics that measure how well predictions match true labels. Below are the main metrics used, defined in the multiclass setting [57].

* **Accuracy**:
  + The proportion of correct predictions over the total number of predictions (Equation 1):
    - ( 1 )
* **Precision (per class)**:
  + For a given class , the number of times it was correctly predicted over the number of times it was predicted (Equation 2 ):
    - ( 2 )
    - where is the number of true positives for class , and is the number of times class was predicted incorrectly.
* **Recall (per class)**:
  + For a given class , the number of times it was correctly predicted over the number of times it was the true labe (Equation 3):
    - ( 3 )
    - where is the number of times class was the correct label but not predicted
* **Macro-Averaged Precision and Recall**:
  + Average over all the classes of the per class precision, recall (Equation 4 and 5, respectively):
    - ( 4 )
    - ( 5 )
    - where is the number of classes.
* **F1 Score (per class)**:
  + The harmonic mean of precision and recall for class (Equation 6):
    - ( 6 )
* **Macro F1 Score**:
  + The average F1 score across all classes (Equation 7):

* + - ( 7 )
* **Top-2 Accuracy**:
  + The proportion of times the true class is among the top 2 predicted classes (Equation 8):
    - ( 8 )
* **Z-score of Accuracy**:
  + Compares observed accuracy to what would be expected under random guessing.
  + Let be the number of classes, the number of samples, and the observed accuracy:
    - Expected accuracy (Equation 9):
      * ( 9 )
    - Variance of accuracy (Equation 10):
      * ( 10 )
    - Z-score (Equation 11):
      * ( 11 )

These metrics provide different insights into how well a classification model performs, with the z-score being particularly useful to compare models across different numbers of classes.

# METHODOLOGY

## Methodology Overview

Inspired by the description of the distinctive features of MSL by Cruz Aldrete et al. [10], we base our methodology on the idea that, at the word level, MSL signs can be divided into 6 minor units:

* Hand Configuration
* Place of Articulation
* Hand Motion Type
* Direction of Hand Motion
* Hand Orientation
* Non-Manual Cues

We attempt to materialize these units as vectorial information by extracting per-frame coordinates for a group of body landmarks, primarily in the hands, from our alphanumeric sign videos. From this core landmark dataset, we create multiple alternative versions:

* Raw landmarks
* Geometrically transformed (to distill the minor units of the sign)
* Dimensionality reduced (using a wide array of reduction techniques)
* Transformed + reduced

We then perform a series of screening evaluations to select the most promising dataset variants, for which we train and benchmark a suite of BERT-based transformer architectures to identify the optimal dataset–model pairing. Once the best combination is determined, we fine-tune the selected model and integrate it into our live-inference pipeline for real-time MSL recognition.

In this chapter, we first describe the raw dataset. Next, we detail our multi-phase data preparation process, including landmark extraction, optional geometric transformations, and optional dimensionality reduction, after which we obtain a wide range of datasets to train models. We then outline a two-stage pruning strategy to select the most promising dataset and model architecture before presenting our best model training and tuning pairing. A diagram summarizing our methodology can be seen in Figure 3.

|  |
| --- |
| Figure 5: Methodology overview flowchart |

## Raw Dataset

Table 4: Samples from the alphanumeric MSL dataset [46], [47]

|  |  |
| --- | --- |
|  |  |
|  |  |

We accessed a publicly shared MSL dataset of 11 signers, each performing 28 letters (those in the English alphabet as well as “LL” and “RR”), and 11 numerals (0–10) approximately ten times—five repetitions per hand—for a total of 4,156 short video clips [9], [45], [46], [47]. The recordings isolate single static and dynamic signs under controlled conditions, providing balanced coverage across signers, classes, and handedness. Table 4 shows 4 frames from video samples of our dataset for the letters “A”, “B”, “C” and “D” by four different people, two with each hand.

The dataset is exceptionally well labeled and organized, following a clear directory and filename convention. At the top level, each signer’s data sits in a folder named pXX, where XX is a number identifying the signer. Within that, subfolders use the pattern Ciclo\_Y\_5\_Z, where Y denotes the cycle number (which goes from 1 to 5) and Z is “Derecha”  or “Izquierda” for left- or right-hand, indicating the hand which is actively performing the sign. Inside each subfolder, video files are named Ciclo\_Y\_5\_Z\_S.mp4, Y and Z keeping the same meaning, and S indicating the specific letter or numeral. This hierarchy encodes signer ID, cycle, handedness, and class at a glance, simplifying downstream filtering and preparation, analysis and training.

It is also important to note that although the sign label is the “natural” or intended tag, the other annotations can in practice be used as labels. For example, we leverage the handedness annotation to solve a secondary classification problem: active hand detection. Generally speaking, MSL uses both hands to articulate signs. However, all signs in this dataset are single-handed. Therefore, we decided to use the handedness information attached to each video, which effectively indicates the active hand, to train a preliminary model which we also use in our live-inference system to select the active hand amongst those which are detected in an image.

We load each clip into a pandas DataFrame, annotating signer ID, sign class, hand used, and frame index to establish a structured foundation for all preprocessing phases.

## Data Organization and Filtering

|  |
| --- |
| Figure 6: Data organization and filtering flowchart |

We make use of the dataset’s existing directory hierarchy to parse the signer ID, cycle number, handedness, and sign class into DataFrame columns. We then apply three filters to ensure data quality:

* We drop clips with fewer than twelve frames to guarantee sufficient temporal context.
* We enforce class balance. The class for “0” has 30 video samples, which represents less than a third of any other class, so we discard it.
* We remove the “7” class entirely because its handshape is visually indistinguishable from “G” in this dataset’s dialect of MSL (which is the same as the one Cruz Aldrete et al. [10] describe), as can be seen in Table 1. With context, these two signs are generally immediately distinguishable. However, in our isolated sign language farming, keeping both classes separately is likely to cause confusion for our models, and combining them would create a class imbalance. It is worth noting that the class for “0” had the same issue with “O”, as can be seen in Table 1, adding a second reason to discard this class.

These steps yield a clean, balanced set of isolated-sign videos.

## Data Preparation

## Preparation Overview

Data preparation proceeds in three steps. First, we extract frame-level landmarks to capture hand shape, pose, and detection context (we call this phase 1, or PH1). Next, we optionally transform these landmarks to isolate the principal components of signs—hand configuration, hand orientation, hand position relative to the body, and hand movement (we call this phase 2, or PH2). Finally, we optionally apply dimensionality reduction (PCA or UMAP with varying component counts) to condense high-dimensional features (phase 3, PH3). Each step generates dataset forks that feed into our evaluation and pruning pipeline in the next stage of our development.

|  |
| --- |
| Figure 7: Data preparation |

## (PH1) Video to Landmarks

## Purpose of Using Landmarks instead of Images

When looking at the hand, signs in MSL decompose into five principal components [10]:

* hand configuration
* hand orientation
* type of hand motion
* direction of hand motion
* hand placement

These are sometimes supplemented by facial or torso cues for semantic meaning [10].

Extracting landmark coordinates offers a direct, low-dimensional representation of these components. Landmarks reduce data size by orders of magnitude compared to raw pixels, eliminate background clutter, and allow us to inherit robustness from MediaPipe’s training on diverse synthetic and real-world hand datasets.

## Basic Use of MediaPipe

MediaPipe Hands and Pose provide real-time hand and body landmark detection from a single RGB frame without the need for specialized hardware [28], [27]. MediaPipe Hands returns 21 three-dimensional hand landmarks, which can be seen in Figure 8, as well as handedness labels, and confidence scores; MediaPipe Pose returns a well-dispersed set of pose landmarks, which can be seen in Figure 9.

|  |  |
| --- | --- |
| Figure 8: MediaPipe hand landmarks | Figure 9: MediaPipe pose landmarks |

Integrating MediaPipe at the beginning of our system's pipeline allows us to inherit from its robust capabilities. MediaPipe Hands was trained on a wide variety of data, with varying lighting, quality of image, camera angles, etc. [27]. Additionally, the model was partially trained on computer generated 3D images of hands [27], which allow it to infer depth coordinates even without using specialized cameras like Microsoft Kinect. Lastly, the lightweight, optimized pipeline ensures frame-level feature extraction runs extremely quickly on CPU (Central Processing Unit)—critical for live inference across different hardware devices.

## Use of MediaPipe for this Project



|  |
| --- |
| Figure 10: Phase 1: landmark extraction |

A diagram summarizing this section can be seen in Figure 10.

We read each video using Python's OpenCV [58] package, and sample the 12 central frames. This strikes a balance between dataset size -- allowing us to obtain over 1000 frames per class -- and temporal coverage of the sign, particularly necessary in the case of dynamic signs. Additionally, it helps address a leakage issue with some of the videos in the dataset where the start (or end, respectively) of the video shows part of the previous (or following, respectively) sign which the signer articulated. Through random exploration of the raw dataset, we found that 12 frames per video generally covered the totality of the sign's action. It is worth mentioning that the author of the paper which presented the dataset [46], [47] used for this project, M. E. Rodriguez, followed the creation of that dataset with a project using it [45] for sign language recognition where 30 frames per sign were obtained, and in cases where the video was too short, the last frame of the video was duplicated until the 30 mark was attained.

For each frame, MediaPipe Hands first detects potential hands. Then for each of these hands, it extracts 21 landmarks, the detected handedness, and a confidence score, as well as other information we don’t use here.

There is often more than one hand per frame. As mentioned previously, the signs in our dataset are all single-handed. Therefore, of the detected hands, one is often "inactive" and provides no information on the sign being performed, and we decided it would be better to train a preliminary simpler model for "active hand detection" which we use prior to the sign recognition model in our live-inference system. To train this model, we need a label for each frame that tells us whether the hand is active or not. The frames or videos are not tagged in this manner directly, but the handedness tags on the video combined with the detected handedness tag and detected handedness confidence score provided by MediaPipe for each frame offer a proxy to “hand activity” which we will describe shortly.

Similarly to the Hands framework, MediaPipe Pose extracts body landmarks, of which we use three (indices 0, 11, 12 in Figure 9). Because there is only one person per video in our dataset, we never obtain more than one detection per frame. However, this also means that our final system only works with one person in the frame. If more than one person were detected, we have no natural way of selecting among them, other than through MediaPipe’s confidence scores potentially, which has no actual relation to our problem.

We omit facial landmarks in this phase because expressions do not influence number and letter signs, and facial expression recognition is a challenging problem in its own right that would extend the scope of this project too much. While this decision simplifies our current pipeline, it limits the potential for extending this project to recognition of signs which incorporate facial expressions. It would be desireable in such a case to adapt the framework to integrate facial landmarks. Also, it is worth mentioning that, as can be seen in Table 4, some of the voluntary signers in the dataset are wearing face coverings, which obviously limits this component of sign language. A different dataset would probably be needed if the study of facial expressions in sign language were a priority.

In total, for each frame we obtain a varying number of rows in a pandas dataframe: one for each hand detected[[1]](#footnote-1). For each of these rows we apply the following logic:

* if it is the only row corresponding to its frame in its video, then we label that row as corresponding to the active hand
* if it is one row of multiple (there are never more than 2) corresponding to its frame in its video, but it is the only one where the detected handedness matches the tagged handedness for the video, then we label that row as corresponding to the active hand; we tag the other rows corresponding to that frame of that video as inactive
* if it is one row of multiple corresponding to its frame in its video, and it is not the only one for which the detected handedness corresponds to the tagged handedness of the video, then we sort the rows in that group by handedness confidence score, and label the one with the highest score as active and the rest as inactive

In this way we obtain a secondary label for each row indicating the activity of the detected hand.

It is important to note that because the data for inactive hands is not relevant for sign recognition, we will drop the rows labeled as inactive right before training sign recognition models. Also, at this point the unit of data in our dataset is a frame. We intend on taking advantage of the sequential aspect of videos for sign recognition, which should be especially useful for dynamic signs. For that, we will also reformat our data right before training so that the rows corresponding to the 12 frames of a single video are concatenated into a single row per video. We do these two dataset transformations right before training instead of earlier to avoid duplicated storage of data and application of data preparation processes.

## PH1 Output Description

The result of this first data preparation process is a pandas dataframe with 69,572 rows and 80 columns, namely:

* 4 video tag columns
  + fileid
  + person\_id
  + cycle\_num
  + handedness
* 3 label columns
  + class\_name and class\_numeric, which indicate the sign
  + active\_hand
* One column to later concatenate rows coming from a single video in the correct order if needed
  + current\_frame
* 21 hand landmark triplet columns
  + h{i}x, h{i}y, h{i}z for i in [0, ..., 20]
* 3 pose landmark triplet columns
  + p{i}x, p{i}y, p{i}z for i in [0, 11, 12]

## (PH2) Geometric Landmark Transformations

|  |
| --- |
| Figure 11: Phase 2: geometric landmark transformations |

As mentioned previously, as far as the hand goes, MSL signs consist of 5 core components—hand configuration, hand orientation, hand placement, hand movement type, and hand movement direction [10]—all of which we already encode via raw landmark coordinates. However, these components intermix in the original camera frame. In PH2, we seek to isolate each component.The diagram in

Figure 11 summarizes the following descriptions, and a more detailed description can be found in Appendix A as a step-by-step pseudocode implementation.

To obtain a representation for the hand orientation, we approximate the palm of the hand to a plane, obtain a pair of vectors which define it, compute a vector normal to them which points the way the palm faces and keep the coordinates for these three vectors.

To obtain a representation for the hand configuration, we use these three vectors and the coordinates of the wrist to define a new frame of reference in which we obtain the coordinates for each of the 21 hand landmarks.

To obtain the hand's position relative to the body, we apply a process similar to the previous two, but using the 3 pose landmarks we extracted earlier to define a plane, obtain a vector normal to that plane, using this one and two which define a plane as a base for a new frame of reference with the nose as the origin. Then we compute the mean of the coordinates of the hand in the original frame of reference, and move this to our new body-centric frame of reference.

Finally, time-based movement components emerge naturally from the other 3 components later when we aggregate per video.

## PH2 Output Description

As mentioned earlier, this phase of data preparation is "optional", meaning that we will keep both this version of the data with the geometric transformation process, and the version of the data without it to train models. The data that does go through this process exits as a pandas dataframe of 69572 rows and 83 columns, namely:

* 4 video tag columns
  + fileid
  + person\_id
  + cycle\_num
  + handedness
* 3 label columns
  + class\_name and class\_numeric, which indicate the sign
  + active\_hand
* One column to later concatenate rows coming from a single video in the correct order if needed
  + current\_frame
* triplet columns for each of the three vectors of the base of the "wrist's frame of reference"

###### h\_v{i}x, h\_v{i}y, h\_v{i}z, for i in [1, 2, 3]

###### 21 hand landmark triplet columns in the wrist's frame of reference

###### wh{i}x, wh{i}y, wh{i}z for i in [0, ..., 20]

###### 1 triplet of columns for the mean hand's coordinate in the "body's frame of reference”

###### cp\_h\_mean\_x, cp\_h\_mean\_y, cp\_h\_mean\_z

###### Note that we no longer keep any pose landmark data.

## (PH3) Dimensionality Reduction

This phase of data preparation explores reducing feature dimensions via PCA and UMAP. We initially looked into this option early in this projects development when we still weren't using BERT based models, but rather KNN models, which struggle to balance high-dimensional data. However, it proved to be a competitive process in our final pipeline, and also provided us a way to analyze the quality of our data through 3-dimensional visualizations of reduced data, amongst other analyses.

Figure [8] shows an overview of the actions performed in this phase. It begins with the output of either PH1 or PH2, which is optional. We then standardize all feature columns—by excluding video tags, labels, and frame indices—to zero mean and unit variance. This normalization ensures that each dimension contributes equally to downstream calculations and mitigates scale disparities.

Next, we systematically apply dimensionality reduction across multiple reducer–kernel combinations and component counts. We test PCA and UMAP, as well as PCA with polynomial, RBF (Radial Basis Function), sigmoid, and cosine kernels, for reduction component counts going from 1 to 15. We selected this range as roughly a quarter of our ∼ 70 landmark features, balancing information retention with computational cost. Our choice of reduction techniques is due to their mature, easy-to-use implementations in scikit-learn and umap-learn, and their potential for on-device real-time use.

|  |
| --- |
| Figure 12: PH3: dimensionality reduction |

##### Using kernels prior to the PCA reducer proved to slow the process down significantly. Computing the full kernel matrix requires evaluating pairwise similarities for every sample in the dataset—an operation whose time and memory requirements scale quadratically with the number of points. To overcome this bottleneck, we turned to the scikit-learn’s implementation of Nyström approximation: rather than forming the complete  kernel matrix, a randomly sampled subset of  “landmark” points is selected, with which only the  and  kernel blocks are computed, and then these are used to reconstruct an approximation of the full matrix. This approximation reduces both computation and storage from to . We used 500 as our value number of landmarks for the approximation, as it greatly reduced the time while maintaining visually similar results in the 3 dimensional case.

##### Each (reducer, kernel, ) combination yields a new dataset fork of the same 69572 rows and columns, where corresponds to the same video tagging, labeling and frame indexing columns as before, and is the number of reduction components.



## 3-Component Reduction Visualization Insights

**We obtained 36 3-dimensional visualizations of our data using our 6 different reducer-kernel combinations, 18 for the data using the geometric transformation processes, and 18 for the data that didn’t. For each of the data preparation processes, we visualized 3 different versions:**

* **the first corresponds to the full per-frame data including rows for both active and inactive hands, and we colored by the active\_hand column**
* **the second corresponds to per-frame data that only includes rows corresponding to active hands, and we colored by the sign class**
* **the third corresponds to per-video data, that also only includes data corresponding to active hands, and we also colored by the sign class**

**The complete set of the 36 visualizations can be seen in Appendix B, however we highlight some of the more insightful ones here.**

**The visualizations colored by sign are much harder to interpret simply due to the number of classes, which makes the variations in color between classes smaller. Of these, the best results were observed using a PCA reducer with a cosine kernel. Still, conclusive results could not be extracted from these due to the slow-changing gradient of colors.** Figure 13 **and** Figure 14 **show the results for the per-frame data with or without the PH2 transformations, respectively.**

|  |  |
| --- | --- |
| Figure 13: 3-component reduction of PH2-processed per-frame data using PCA with a cosine kernel and coloring by sign | Figure 14: 3-component reduction of non-PH2-processed per-frame data using PCA with a cosine kernel and coloring by sign |

**However, the visualizations using the active\_hand label, which only takes two values, are much easier to interpret.**

**In** Table 5**, we see a comparison of the PH2-processed per-frame data using PCA with our 4 different kernel options. Of these 4, the cosine and RBF kernels stand out as the best.**

Table 5: Comparison of PH2-processed per-frame data using PCA and 4 different kernels

|  |  |
| --- | --- |
| Figure 15: Using a cosine kernel | Figure 16: Using a polynomial kernel |
| Figure 17: Using an RBF kernel | Figure 18: Using a sigmoid kernel |

**In** Table 6**, we see the same comparison for the non-PH2-processed per frame data.**

**In this case, the cosine kernel seems to more clearly take the lead.**

Table 6: Comparison of non-PH2-processed per-frame data using PCA and 4 different kernels

|  |  |
| --- | --- |
| Figure 19: Using a cosine kernel | Figure 20: Using a polynomial kernel |
| Figure 21: Using an RBF kernel | Figure 22: Using a sigmoid kernel |

**The “axial” nature of the RBF reductions is interesting. It seems the reduction has revealed that the data follows some variables for which we see a somewhat continuous increment across our dataset, even though it might not match our classes. In fact, if we look at the PCA reductions using the RBF kernel for the per-frame data corresponding only to active hands with or without the PH2 processing, as we see in** Figure 23 **and** Figure 24**,we will notice that the axial is still present, even though it does not separate between our classes at all. One theory we have is that it might correspond to the time-aspect of our per-frame data. These reductions were not in anyway “aware” of the frame index for the data, however if we look at the reductions for the per-video data using the same reduction techniques in** Figure 25 **and** Figure 26**, we can see that the axial nature is “lost”. Intuitively, time being the root of these continuous progressions along lines is an appealing interpretation, though it is entirely possible that the feature with the linear characteristic is entirely unrelated.**

|  |  |
| --- | --- |
| Figure 23: 3-component reduction of the PH2-processed per-frame data using PCA with an RBF kernel and coloring by sign | Figure 24: 3-component reduction of the non-PH2-processed per-frame data using PCA with an RBF kernel and coloring by sign |

**As far as comparing along the preprocessing techniques outside of dimensionality reduction, for the use or lack of use of PH2 geometric transformations, there is no conclusive evidence giving an advantage to either side, as we can see in Appendix B. Especially for the sign colored graphics, it is hard to notice any true separation of the data by classes, with any combination of processes. For the active\_hand colored graphics, PH2-processed data seems to be slightly more separable if we look overall, but if we look only at the best technique (PCA with a cosine kernel), using PH2 processing seems to have an impact, but doesn’t notably help with the separation of classes.**

|  |  |
| --- | --- |
| Figure 25: 3-component reduction of the PH2-processed per-video data using PCA with an RBF kernel and coloring by sign | Figure 26: 3-component reduction of the non-PH2-processed per-video data using PCA with an RBF kernel and coloring by sign |

##### **On a separate issue, the impact of the of per-frame or per-video data is difficult to compare. The per-video data has 12 times less samples, so the task is not very straightforward. For most cases, we could not observe a notable difference in the quality of separation of data. However, besides the case of RBF kernels, there were a few other cases where there was a noticeable difference in the general shape that the reduced data formed. The most remarkable is shown in** Figure 27 **and** Figure 28**, where PH2-processed data is reduced using UMAP. The reduction for the per-frame data shows similar results to what we generally observed when using UMAP: a central clump of data points closely surrounded by many small clumps. More examples like this can be seen in Appendix B. However, for the per-video data we can see two clearly separated clumps. Again, because of the number of classes, at first sight it is hard to interpret whether or not this separation has any relation to our classes.**

|  |  |
| --- | --- |
| Figure 27: 3-component reduction of the PH2-processed per-frame data using UMAP and coloring by sign | Figure 28: 3-component reduction of the PH2-processed per-video data using UMAP and coloring by sign |

## Dataset and Model Selection

## Goal

We aim to use transformer encoder models for real-time MSL recognition, while also keeping both delay and computational cost low. Our goal is to interpret continuous signing with minimal lag, while running directly on standard devices. For this, we focus on five lightweight models—bert-tiny, bert-mini, bert-small, bert-medium, and DistilBERT—whose sizes are compared in Table 7. We use pre-trained models which are based on the architectures described in “Well-Read Students Learn Better: On the Importance of Pre-training Compact Models” [37] and “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter” [38]. We use a structured pruning approach to explore which dataset-model combinations work best, without training every possible pair.

|  |  |  |  |
| --- | --- | --- | --- |
| Encoder | Number of Layers | Hidden Dimension size | Number of parameters |
| bert-tiny [59] | 2 | 128 | 4.43 million |
| bert-mini [60] | 4 | 256 | 11.3 million |
| bert-small [61] | 4 | 512 | 29.1 million |
| bert-medium [62] | 8 | 512 | 41.7 million |
| DistilBERT [63] | 6 | 768 | 66 million |

Table 7: Size comparison of different BERT inspired architectures (see [59])

###### As our dataset preparation pipeline can generate hundreds of variants—each defined by different combinations of preprocessing steps, dimensionality reducers, and data‐unit formats—pairing every one of these with each of our five candidate encoder architectures would quickly become both too computationally and time consuming for this project.

###### To address this, we implement a two-stage pruning strategy that filters both datasets and models before committing to full training. First, we employ lightweight proxy evaluations—based on KNN models—to identify the most promising preprocessing configurations without training the encoders. Once the top dataset variants are determined, we then perform a second screening across our encoder architectures using a reduced number of epochs. Finally, we explore the best architecture with a select group of datasets more in depth to choose the dataset-model pair we will focus all our resources in the next stage of development. With this two-step approach we are able to focus our resources on the dataset-model configurations that appear to have the most potential. Figure 29 summarizes this process.

|  |
| --- |
| Figure 29: Dataset and model selection |

## Dataset Selection

As mentioned in the previous section, the challenge at hand is reducing the number of candidate dataset-model combinations. We began with 546 dataset variants generated by combining:

* PH2 (2 options)
  + Applied
  + Not applied
* PH3 options (91 total):
  + PCA for components, with going from 1 to 15, making for 15 options
  + PCA and one of four kernels (Polynomial, Cosine, Sigmoid or RBF) for components, with going from 1 to 15, making for options
  + UMAP for components, with going from 1 to 15, making for 15 options
  + PH3 not applied (1 option)
* Data-unit formats (3):
  + all rows
  + active-hand rows per frame
  + active-hand rows aggregated per video

As we can see, the largest factor here is the number of PH3 options. Therefore, we first focused on finding what the best combinations in that area of choices would be, and discarding the rest. Figure 30 shows the strategy used to do so.

|  |
| --- |
| Figure 30: Dataset Selection |

## Selection of Subsets Using K-Means

To have a more robust interpretation of the value of the different data preparation processes, we wanted to evaluate their impact as the number of classes grew. To do so, we decided to train KNN models on each of the datasets generated throughout the data preparation phases, but to do so on filtered versions of those datasets where only rows corresponding to certain subsets of all our classes were kept.

To select the subsets of classes we would filter our datasets with, we decided to use the data generated directly after landmark extraction with MediaPipe, and use a K-Means [64] model to obtain the centroid for each class. Then, for each possible subset of classes, we computed a difficulty score based on the cosine similarity between the centroids corresponding to the classes in that subset. Our assumption is that the closer the centroids were, the more difficult they should be to differentiate. Finally, for each number of classes going from 2 to the total number of classes (37), we sorted all of the subsets for that number of classes by their difficulty score, and chose 3 from those: one “easy”, which had the difficulty score closest to the 10th percentile among candidate subsets; one “average”, which had the difficulty score closest to the 50th percentile among candidate subsets; and one “hard”, which had the difficulty score closest to the 90th percentile among candidate subsets.

The description above was a simplification of the process. We actually did not compute difficulty scores for every possible subset of classes due to computational limitations. Instead, for each number of classes from 2 to 37, we randomly sampled across all the candidate subsets at most 1000 of them, and computed the difficulty scores for those.

## Fitting K-Nearest Neighbors Models for each Subset, for each Dataset Candidate

Originally, our intention was that now that we had the three selected subsets for each number of classes, we would take each of the 546 datasets, and filtered them for each of the subsets, and fit several KNN models to the data. However, once again to be computationally resource-effecient, we only used the subsets corresponding to a reduced number of options, specifically the odd numbers going from 3 to 37 (our total number of classes).

For each the datasets, we fit KNN models with the number of neighbors being amongst 1,3,5 and 7.

Then we computed the accuracy of each model. Because we were trying to compare models’ quality across different numbers of classes, comparing models’ metrics directly gave an advantage to models trained on lower numbers of classes.

To compensate for this, we decided to compute a significance score for the observed accuracy for each of the models, and compare them that way. We did this by considering that a score was more significant the better it was when comparing it to what random guessing would result in for the number of classes in question.

## Computing Significance Scores based on Observed Accuracy

Let’s first recall the definition of multiclass accuracy. Given an -class problem with total samples, let be the number of samples whose true label is and whose predicted label is . Then Equation 12 shows a relation between the concerned numbers of samples.

( 12 )

Based on Equation 12, we can specify the formula for accuracy given in Equation 1 as the one shown in Equation 13.

( 13 )

We assume two things under our random‐guessing model:

1. **Uniformly random true labels.** The dataset has basically no class imbalance, so we make the assumption that each class  is equally likely to be the true label (Equation 14).

for ( 14 )

1. **Uniformly random, independent guesses.** The classifier has no information and picks each class  with equal probability, independently of the true label (Equation 15).

for ( 15 )

Because we treat “true label” and “guess” as independent under this model, the joint probability of any specific ordered pair is the product of the two individual probabilities (Equation 16).

( 16 )

Now, assuming uniform random guessing among classes:

1. Each ordered pair has probability



1. Correct predictions occur when . There are such diagonal events, each of probability , so the probability of a correct prediction is , as seen in Equation 17.

( 17 )

Therefore, the expected (mean) accuracy under random guessing is shown in Equation 18.



( 18 )

Under random guessing, each of the examples has probability of being correctly classified. We model the total number of correct predictions, , as a Binomial(, ) random variable.

1. The expected value of is: ( 19 )
2. The Variance of is: ( 20 )
3. And we have that the accuracy is , therefore the variance of is as shown in Equation 21.

( 21 )

1. Thus, the standard deviation is as shown in Equation 22.

( 22 )

1. And finally, for an observed accuracy , the Z-score is as shown in Equation 23.

( 23 )

## Comparing Models across Number of Classes

Now that we had significance scores for each of the models across number of classes we could compare them more fairly.

|  |
| --- |
| Figure 31: Mean of the top 10 significance scores for each number of classes studied |

The first result to point out, which contextualizes the rest, is seen in Figure 31. The significance scores seem to linearly grow as the number of classes does. Knowing that the dataset is evenly balanced across classes, this means that the number of samples per dataset is directly proportional to the number of classes. This has a two-fold benefit for the models trained on more classes:

* The formula for the z-score we showed above is directly proportional to . However, this is somewhat contradictory to the linear growth we are extrapolating from these 18 datapoints, so there must be at least some other factor involved.
* Models generally become better the more data we have to train them on. Although we have no precise theoretical relation between these two variables, it is possible that this an important factor in the linear trend we observed in Figure 31.

Additionnally, as we know an increase in number of classes is also directly related to a diminishing accuracy—more classes means more options for the classifier to predict incorrectly. The fact that the number of samples’ positive effect seems to be outweighing the number of classes negative effect in a consistent, linear fashion suggests that our model could continue to obtain even more significant results with more data. Therefore, if in the future larger datasets of similar quality were made available for MSL, it seems the methodology explored here should scale well.

|  |
| --- |
| Figure 32: Evolution over the number of classes of the significance scores obtained for data trained on per-frame (blue) or per-video (orange) data |

A phenomenon which we believe could share an explanation is observed for models trained on per-frame data vs on per-video data. In Figure 32, we see that both curves showing the evolution over the number of classes appear to grow linearly whether we use per-frame or per-video data, but that the per-frame curve grows at with a significantly larger rate. We believe this to be for a few reasons:

* As we mentioned before, the per-video data, because of how it is constructed, has one twelfth of the number of samples of the per-frame data. This fact combined with the general trend we saw for number of classes suggests to us that once again there is a significant impact from the number of samples in the significance scores we observe
* Per video-data will have a much higher dimension count (here again: a factor of 12). This will be the case even for the data that went through dimensionality reduction because we do the “concatenation” of rows corresponding to a single video right before fitting the models, meaning after the dimensionality reduction occurs. High dimensionality is something KNN models can sometimes struggle with, as they have no intrinsic capability to favor certain features over others, and more features mean more opportunities for “distraction”
* Additionally, KNN models have no concept of sequences or time. The feature columns have an intrinsic structure which relates them as data from frames in videos, but this is lost for KNN, therefore it just adds complexity with no structural gain

|  |
| --- |
| Figure 33: Evolution over the number of classes of the significance scores obtained for data passed through PH2 (orange) vs data that did not (blue) |

Next, we studied the impact of our data preparation processes. In Figure 33, we compare data that went through geometric transformations with data that did not. Somewhat alarmingly, we see that PH2 seems to have a negative impact on our results, at least for KNN models. While PH2 does cause a rise in the number of feature columns, it is minimal (75 vs 72). Therefore, the early interpretation we take from this is that some valuable information about MSL may be being lost in our transformations.

Even so, we choose to continue exploring the use of PH2 throughout the rest of our experiments. The aim of this is not only to obtain the most accurate or robust sign language recognition system, but also to test the hypothesis that signs can be morphologically decomposed into the six components described by Cruz Aldrete et al. [10]. While the drop in performance raises valid concerns, continuing to evaluate PH2 allows us to examine whether the issue lies in the implementation or in the assumption of separability itself. In this sense, PH2 functions as both a preprocessing strategy and a tool for investigating the learnability of linguistically motivated structure from landmark data.

The next aspect we study is the impact of dimensionality reduction. We do so at a more extensive level then the previous aspects because, as we mentioned before, a large majority of the decisions which are causing the explosion in the number of dataset options we have can be grouped under the umbrella of dimensionality reduction. It is our first priority in terms of option pruning, and we make decisions based on the results we obtain in this phase of analysis.

The first result we look into is seen in Figure 34, which shows that no substantial loss or gain in the significance of accuracy results from using data reduction techniques when using KNN models. This is a promising sign, as it would somewhat help speed up training of neural network models if it held true later in the process. However, it is worth noting once again that KNN models struggle with high-dimensional data, and this is not the case for all models. In fact the opposite is sometimes true. So we might not see the same benefits (or lack of adverse effects) when using these same techniques with other architectures.

|  |
| --- |
| Figure 34: Evolution over the number of classes of the significance scores obtained for data passed through PH3 (orange) vs data that did not (blue) |

|  |
| --- |
| Figure 35: Evolution over the number of classes of the significance scores obtained using different reduction techniques |

The next option we evaluate is the reducer choices. As we can see in Figure 35, using no reducer, PCA, PCA combined with a kernel, or UMAP does not produce drastically different results. However, UMAP performs slightly worse as the number of classes grows, and due to the fact that it is also slower and might affect live-inference in some cases, we decide that moving forward we will no longer investigate it.

As to why PCA seems to perform better than UMAP, it is difficult to conclude. In the abstract, the main difference between the two is that PCA is a linear projection method that identifies directions of maximum variance in the data, with the goal of preserving global Euclidean geometry using as few dimensions as possible. UMAP, on the other hand, is a nonlinear graph-based embedding method that constructs a local neighborhood graph and optimizes the low-dimensional layout to preserve those local relationships, with the goal of capturing the underlying manifold structure. The other two components we must consider to draw a conclusion from these theoretical differences between reducers at this point of our analysis are the model and the data.

KNN is a model based on proximity in the input space—it assumes that nearby examples are likely to belong to the same class, and it makes predictions by querying the majority class of a sample’s nearest neighbors. It is possible that this aspect somehow favors PCA, due to the fact that PCA preserves global distances and relative positioning more consistently, making KNN neighborhoods more stable after projection. Nevertheless, this in theory should be counterbalanced and even overwhelmed by UMAP’s focus on preserving local relationships, which could help maintain neighborhood structure even if global distances are distorted.

Because of this, we turn to our data for a better explanation. As we noted earlier, our best-performing models—those in Figure 35—are based on per-frame data. This makes it unlikely that the reducers are capturing motion over time, and instead suggests that the structure comes from how the landmarks are arranged in each frame.

Each sample is made up of coordinates for the hands and body, in theory taken while a person is articulating a sign. In this context, it's worth noting that the hands often take on clear, repeatable shapes. For example, fingers (especially the non-thumb fingers) are usually either fully extended, fully retracted, or somewhere close to halfway between. This pattern, noticeable in Figure 1 and Figure 2, means that across the dataset, the same finger may show up in just a few typical positions.

If that's the case, then for each finger, the variation across samples might fall into three clusters: extended, curved halfway, and retracted. These clusters might not lie along a perfect line for all coordinates, but could still form a simple structure in space—something like a triangle or a curve. Even if it's not strictly linear, this kind of pattern could still be captured well with one or two PCA components. UMAP, by contrast, is designed to preserve more complex local relationships, which may not be necessary in this setting and could even introduce distortions.

|  |
| --- |
| Figure 36: Evolution over the number of classes of the significance scores obtained combining different kernels with PCA |

Following that, we study the different kernels we can use to combine with PCA. Figure 36 shows that the best results come from standard PCA, followed closely by the cosine kernel. This suggests that most of the variation relevant to class separability is captured by global, linear structure in the landmark data — particularly by the overall direction and magnitude of full-pose vectors. The strong performance of the cosine kernel further implies that even without magnitude, orientation alone carries most of the discriminative information. In contrast, kernels like polynomial, RBF, and sigmoid, which introduce nonlinear or locally sensitive distortions, consistently degraded performance. This supports the interpretation that sign classes differ in broad, structured ways, rather than through complex feature interactions or small local differences.

From this point forth, in the interest of speeding up our development, we decide to continue with only the most promising reducer-kernel pairing: PCA with no kernel.

Having selected our reduction technique, we studied what the best number of components would be. At this point, we focused on the case for the maximum number of classes, and therefore looked at the accuracy score directly instead of the significance score. Figure 37 shows that PCA generally performed better as the number of components went up.

Having selected PCA as our dimensionality reduction technique, we next studied how many components to retain. For this, we focused on the case with the maximum number of classes (37) and plotted the top-10 mean classification accuracy for different component counts. As shown in Figure 37, accuracy increased steadily with the number of components, especially between 1 and 10. This indicates that most of the information relevant to classification is captured within the first few PCA components. After around 10 components, performance gains became much smaller, suggesting diminishing returns.

|  |
| --- |
| Figure 37: Mean of the top 10 accuracy scores for different number of components in a PCA reducer |

To better understand what information is preserved or lost during dimensionality reduction, we also examined the explained variance ratio for each PCA component, both with and without the PH2 transformation, for the per-frame data. Figure 38 and Figure 39 show Scree plots for both cases. In both, the first few components capture most of the variance—over 70% by the third component, and over 90% by the sixth. However, the PH2 version shows a slightly steeper drop-off, with later components contributing less than in the original version. This suggests that PH2 preserves the broadest geometric trends but may suppress finer-grained distinctions, potentially reducing the model’s ability to separate closely related signs.

Importantly, PH2 does retain both hand configuration and placement information, but encodes them separately: configuration is captured in a hand-centric frame of reference, while placement is represented by the mean hand position in a body-centric frame. This structure preserves the key elements required to describe a sign, but assumes that these components can be treated independently. The sharper drop in explained variance after the first few PCA components suggests that some of the interaction between these elements — for example, how handshape might vary depending on its location — may be lost in this decomposition. This loss of interdependence could explain the reduction in classification performance when using PH2-transformed data. Even so, as we mentioned previously, we continue to explore this option in the interest of thouroughly evaluating the hypothesis of separability of the 6 morphological components described by Cruz Aldrete [10].

|  |
| --- |
| Figure 38: Scree Plot: Variance Explained by PCA Components (non-geometrically transformed data) |

|  |
| --- |
| Figure 39: Scree Plot: Variance Explained by PCA Components (geometrically transformed data) |

## Active Hand Detection Model

In addition to this dataset reduction, we also obtained an excellent active-hand prediction model during exploration using the following configurations:

* Use PH2 geometric transformations
* Use PH3 dimensionality reduction, with 15-component, no kernel, PCA reducer
* Use 5 neighbors

In fact, we obtained several KNN models that obtained very similar results. Table 8 shows the best results on the testing data for a selection of choices:

Table 8: Active hand detection models best scores

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model configuration | Accuracy | Macro F1score | Macro Precision | Macro Recall |
| No PH2, No PH3 | 0.9790 | 0.9790 | 0.9790 | 0.9790 |
| Yes PH2, no PH3 | 0.9860 | 0.9860 | 0.9861 | 0.9860 |
| No Ph2, yes PH3 | 0.9800 | 0.9800 | 0.9800 | 0.9800 |
| Yes PH2, yes PH3 | 0.9860 | 0.9860 | 0.9861 | 0.9860 |

As we can see, we obtained excellent, almost identical results across the board. Because of this, to make the final live-inference system smoother, we use the data configurations that the sign recognition model requires for the active hand detection model as well.

Going forward, we will no longer be exploring the datasets containing rows for inactive hands, and can further reduce our candidate datasets to 2 × 2 × 2 = 8 final datasets to explore with our 5 candidate model architectures.

## Most accurate KNN Sign Recognition Models

As mentioned previously, the primary architecture explored in this study is the Transformer, specifically some of its BERT variants. In the interest of having a point of comparison for those models with other more traditional ML models, we report in Table 9 the best results we obtained when claissifying the 37 alphanumeric classes using KNN, with the different data preparation combinations we explore.

Table 9: Scores for best KNN alphanumeric sign classifiers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dataset configuration | Accuracy | Macro F1 score | Macro Precision | Macro Recall |
| Per-frame data, no PH2, no PH3 | 0.9407 | 0.9402 | 0.9405 | 0.9405 |
| Per-frame data, no PH2, yes PH3 | 0.9202 | 0.9194 | 0.9199 | 0.9198 |
| Per-frame data, yes PH2, no PH3 | 0.8211 | 0.8196 | 0.8198 | 0.8218 |
| Per-frame data, yes PH2, yes PH3 | 0.7954 | 0.7938 | 0.7944 | 0.7962 |
| Per-video data, no PH2, no PH3 | 0.5992 | 0.5977 | 0.6185 | 0.5991 |
| Per-video data, no PH2, yes PH3 | 0.5621 | 0.5607 | 0.5853 | 0.5630 |
| Per-video data, yes PH2, no PH3 | 0.4994 | 0.4990 | 0.5150 | 0.4997 |
| Per-video data, yes PH2, yes PH3 | 0.5096 | 0.5082 | 0.5211 | 0.5091 |

As expected, Table 9 largely reflects the trends observed throughout our analysis of the dataset selection process. While previously we focused on the average of the top 10 scores , grouping by different metrics, the current results center on the single best performing models. It is at least reassuring that these top-1 results confirm our earlier findings at the highest level:

* Using per video data instead of per frame data drastically worsens performance for the KNN models.
* Using PH2 transformations noticeably worsens performance for the KNN models.

What stands out however, is that PH3, despite having negligible effects on the performance of the models when taking a slightly more comprehensive view in the earlier sections, appears to have a discernible negative impact for the top end of KNN models. That being said, we refrain from drawing strong conclusions from this.

## Model Selection

We fine-tune each of the 8 datasets with all 5 encoders for a moderate epoch budget approximately proportional to the number of parameters per model seen in Table 7.

For all neural networks in this section and going forward, we use the following training configurations:

* Optimizer: AdamW [65]
* Learning rate:
* Loss function: Cross Entropy Loss [66]

We selected these parameters after admittedly mild exploration, but following the expert advice of Víctor Hugo Martínez Sánchez.

Concerning the number of epochs, we attempted to compensate for certain dataset or model, respectively, size differences by giving additional training epochs to models trained on smaller datasets or using lighter architectures, respectively. We used a system that allocated epochs roughly proportional to the size of the model, and the number of rows in the dataset.

At this point of the process, to accelarate the exploration, we used data loaders with a batch size of 1024. This turned out to be too aggressive, especially for the per-video models that have fewer samples, and negatively affected our training. We believe this can be explained by the model having fewer opportunities to “apply what it has learned” in each epoch. This is because a higher batch size means fewer batches per epoch, and back propagation occurs once per batch. Thus, in the later parts of our explorations, we will lower the batch size substantially.

|  |
| --- |
| Figure 40: Mean of the top 3 accuracies by loaded model |

For each combination of model and dataset, we log accuracy, top 2 accuracy (meaning the rate at which the true label appears in the top 2 likeliest classes according to the model), macro precision, macro recall and macro F1 score.

As we can see in Figure 40, the bert-mini architecture [60] previewed the best results in this exploration, and moving forward we discard the study of all other architectures.

|  |
| --- |
| Figure 41: Mean of the top 3 accuracies by unit of data. Blue: per-frame; orange: per-video. |

We also conducted some minor analysis of other configuration parameters.

As we can see in Figure 41, once again we obtained substantially better results for per-frame data than for per-video data, even though Transformer architectures naturally incorporate the sequential aspect of videos through positional encodings. This is a surprising result, which could be explained by:

* The low number of epochs used. However, we added a factor to try to compensate for this, and allowed for additional training for the per video data. Additionally, we will continue to observe these results as we continue to augment the number of epochs in the latter stages of this study
* An alternate, though untested, reason for this might be that the sequence length (12) for videos is not large enough for the Transformer to benefit from this aspect. Transformers originally became a popular option over RNNs because they performed better at long-range dependency tracking [32]. However, this is because transformers are generally good at tracking *regardless* of the distance between tokens (in our case frames), so this is not a fully satisfactory explanation for this result.

Additionally, we also studied the impact of phases 2 and 3 on our models. Figure 42 shows that while using PH2 processes has a massive negative effect on accuracy, using PH3 processes has very little effect. The first result continues with a line of worrying results. However, as we will see in the Discussion section of this report, this is a reflection of a worrisome phenomenon, but not the expected one.

|  |
| --- |
| Figure 42: Top 3 Mean accuracy for each boolean pair of (PH2, PH3) use as part of data preparation, for all model architectures |

## Final Selection

For our final selection process, we fine-tuned bert-mini models on each of the 8 selected datasets for a varying number of epochs to compensate for different configurations have more or less data, though consistently 5 times more than in the previous phase of exploration. Additionally, we also adjusted the batch size. As mentioned previously, having a very large batch size negatively affected our models, especially in the per-video cases. Thus, we used a batch size of 128 for the per-frame models, and of 16 for the per-video models. For each model, we calculate accuracy, top-2 accuracy, macro F1, macro precision and macro recall scores. Table 10 shows the results we obtained.

Table 10: Final dataset selection scoring metrics

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset configuration | Accuracy | Top-2 accuracy | Macro F1 score | Macro Precision | Macro Recall |
| Per-frame data, no PH2, no PH3 | 0.9670 | 0.9893 | 0.9670 | 0.9670 | 0.9670 |
| Per-frame data, no PH2, yes PH3 | 0.9565 | 0.9839 | 0.9565 | 0.9567 | 0.9566 |
| Per-frame data, yes PH2, no PH3 | 0.9284 | 0.9676 | 0.9285 | 0.9287 | 0.9288 |
| Per-frame data, yes PH2, yes PH3 | 0.8884 | 0.9512 | 0.8884 | 0.8888 | 0.8888 |
| Per-video data, no PH2, no PH3 | 0.8899 | 0.9360 | 0.8894 | 0.8957 | 0.8888 |
| Per-video data, no PH2, yes PH3 | 0.8220 | 0.9040 | 0.8185 | 0.8289 | 0.8219 |
| Per-video data, yes PH2, no PH3 | 0.6248 | 0.7580 | 0.6235 | 0.6334 | 0.6260 |
| Per-video data, yes PH2, yes PH3 | 0.6159 | 0.7606 | 0.6190 | 0.6372 | 0.6166 |

Following these results, we decided to fully invest in the no geometric transformations, no dimensionality reduction, per-frame dataset.

## Selected Model Finetuning

We explored the following candidate hyperparameters for the BERT-mini model:

* Pre-trained model:
  + bert-mini-uncased [60]
* Optimizer:
  + AdamW [65]
* Learning rate:
  + {, , , }
* Weight decay:
  + {, 0}
* Loss function:
  + Cross Entropy Loss [66]
* Number of epochs:
  + {2000, 4000, 5000, 6000, 10000, 12000}
* Batch size:
  + {128, 256, 1024}

Ultimately, we obtained the best accuracy using the following configurations:

* Dataset
  + Data unit: frame
  + PH2 geometric transformations: not applied
  + PH3 dimensionality reduction: not applied
* Model
  + Pre-trained model: bert-mini-uncased [60]
  + Optimizer: AdamW [65]
  + Learning rate:
  + Weight decay: 0
  + Loss function: Cross Entropy Loss [66]
  + Number of epochs: 6000
  + Batch size: 256

# RESULTS AND DISCUSSION

## Results

Using the configurations described in the last section of the previous chapter, we trained and evaluated a model. On our testing dataset, which represented 20% of our complete dataset, we computed the same 5 metrics for our model as before: accuracy, top-2 accuracy, macro F1 score, macro precision and macro recall. We obtained the results seen in Table 11.

Table 11: Metrics for most accurate model in testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Accuracy | Top-2 accuracy | Macro F1 score | Macro Precision | Macro Recall |
| BERT-mini | 0.9740 | 0.9907 | 0.9740 | 0.9742 | 0.9740 |

The training loss was computed in each epoch of the training loop and can be seen in Figure 43.

|  |
| --- |
| Figure 43: Training loss evolution for most accurate model |

After testing, the confusion matrix was computed and can be seeing in Figure 44.

|  |
| --- |
| Figure 44: Confusion matrix for most accurate model |

## Discussion

## Comparing to Similar Studies

This study closely resembles that of Rodriguez et al. [45], even using their dataset as a starting point. Nevertheless, there are few key difference in the raw data each of the studies use. In our case we dropped 2 classes, those of ‘0’ and ‘7’, and train a single model for both static and dynamic classes for the remaining 37 classes. In Rodriguez et al.’s work, two different systems are obtained, each developed with separate portions of the total data: one for static classes, and one for dynamic classes. Then, for each of these, different architectures are explored: GBL and SVM for static signs, and LSTM and GRU for dynamic signs.

To have a frame of reference for the results obtained here, we compare in Table 12 our testing metrics to those of Rodriguez et al.

Table 12: Comparison with Rodriguez et al. [45] study

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Macro F1 score | Macro Precision | Macro Recall |
| (Static signs) GBL [45] | 77.04% | 76.85% | 76.82% | 76.19% |
| (Static signs) SVM [45] | 94.07% | 93.73% | 94.25% | 93.56% |
| (Dynamic signs) LSTM [45] | 79.59% | 80.31% | 80.93% | 80.28% |
| (Dynamic signs) GRU [45] | 84.69% | 86.01% | 86.00% | 85.57% |
| (Present work) (Dynamic and Static signs) BERT-mini | 97.40% | 97.40% | 97.42% | 97.40% |

As we can see, our most accurate model outperformed theirs across the board for the testing data. This is an even more relevant result considering that our model considers more classes (37 vs 29 or 10), and that our system deals with both dynamic and static signs equally, making it a theoretically more challenging problem.

Given that the data preparation process used for our most accurate model does not use either the geometric transformations nor the dimensionality reduction techniques, but only the feature extraction techniques based on MediaPipe’s landmarking framework, and the fact that Rodriguez et al. also use very similar tools in their study, we conclude that these results can be attributed to the following differences in methodology:

* We use Transformer-based models. This architecture has revolutionized ML in general, but also sign language recognition, and our results back this trend. Even though we are not taking advantage of the sequential aspect of videos with our most accurate model (which works on a per-frame basis), this architecture still obtained excellent results.
* Our system incorporates a preliminary model to “pick an active hand”, taking advantage of the single-handed nature of the signs in our dataset. Rodriguez et al take the data for both hands as input for their models, potentially leading to more noise, but also leading the way for a more natural extension of their models to bimanual sign recognition.
* Our system tackles static and dynamic signs in a unified fashion. This intuitively seems like a more challenging problem, but it can also lead to models learning shared features among the two. As we saw in the dataset exploration section, our data proved more fruitful the more classes, and therefore samples, we had. This can be seen a specific example of that phenomenon.

Our approach also includes some other differences, but it is harder for the authors of this study to see any natural positive impact these could be having. For example, our system extracts landmarks not only for the hand, but also for the body. However, this information should not provide much useful information for two reasons:

* As mentioned in the introduction, all the signs in the dataset occur with the hand placed in the same, default area of the body which is the area in front of the chest. This would indicate to us that not much can be gained from knowing where certain key bodyparts are located with relation to the hand
* Additionally, because our most accurate model works on a per-frame basis, it has no concept of evolution over time and cannot infer movement. Therefore, for movement signs for which the hand’s position relative to the body does change (namely “LL” and “RR”), this evolution can not be tracked.

## Interpretation of the Confusion Matrix

The confusion matrix seen in Figure 44 reflects the metrics obtained during testing, with most predictions falling along the diagonal—confirming a strong overall performance. However, the 2.6% error rate is not uniformly distributed, and some patterns help us understand where the model struggles. To better illustrate the points made in this section, Figure 45 and Figure 46 show the alphanumeric signs once again, thanks to the work of IncluSor [15].

|  |  |
| --- | --- |
| Figure 45: MSL alphabet, from [15] | Figure 46: MSL numbers, from [15] |

The most noticeable drops in diagonal density occur for the “LL” and “RR” signs being predicted as “L” and “R”, respectively (with 11 “LL” samples being predicted as “L”, and 12 “RR” samples being predicted as “R”). These dynamic signs are derived from their static counterparts (“L” and “R”) by dragging the same hand shape across the chest, without changing the configuration or orientation, but changing the placement. This is naturally challenging for a classifier that processes data on a per-frame basis, since it lacks any notion of temporal movement. Interestingly, for both of these sign pairs (“L” and “LL”, “R” and “RR”), the classifier tends to favor the static version. This may be due to class imbalance—about 10% more samples exist for the static signs—which could bias the model during training.

However, returning to the issue of dynamic signs for a transformer for which the sequence is entirely made up of one token (frame, in our case), it is interesting to compare how it deals with other signs incorporating movement. Across several examples, it appears that the model has learned to extract some sense of hand orientation and hand configuration directly from the raw landmark data, even without the PH2 transformation.

Starting with hand orientation:

* In the case of the number signs:
  + We observe in Table 1 that the signs can be grouped into three parts based on hand orientation: {1, 2, 3, 4, 5, 9} have the back of the hand facing the camera and the fingers pointing upwards (number 8 in Figure 3); {6, 8} have the back of the hand facing the camera but the fingers pointing inwards to the side (number 9 in Figure 3); 10 starts identically to 5, but then moves to have the palm face the floor with the fingers pointing towards the camera (number 4 in Figure 3). This is visible in Figure 46.
  + We notice that there are several cases of minor confusion for signs having the same hand orientation, but none for signs having different hand configurations (except for one, which we believe can be explained by another phenomenon which we will discuss later).
    - In particular, we see minor confusions between “3” and “4”, as well as confusions where a “9” is predicted to be one of “3”, “4” or “5”. The latter case is more telling, as the “finger retraction” motion [10] used to articulate “9” results in a noticeable difference in hand configuration compared to “3”, “4” and “5”, whereas the hand orientation remains the same.
  + Additionally, we also notice that there is no confusion between “5” and “10”. These two signs share the same configuration, but “10” is articulated by rotating the hand and changing it’s orientation towards the ground.
* In the case of the (“V”, “2”) and (“W”, “3”) pairs:
  + These two pairs are examples of static signs that share the same hand configuration and placement, but where the hand is facing opposite directions. There is basically no pairwise confusion for these, which again indicates the model is extracting information on hand orientation.
* The case of “G” and “L”:
  + Similarly to the previous case, these two static signs share hand placement and configuration, but vary in terms of hand orientation, and no confusion is observed. Additionally, no confusion between “G” and “LL” is observed either. This is a particularly relevant result when considering the confusion between “L” and “LL”, given that the only difference from that case is hand orientation.
* The case of “K” and “P”:
  + These two signs share the same hand configuration, but articulating “K” involves “bobbing the wrist” in a way that affects hand orientation only. Once again, we see basically no confusion between the two classes, revealing the apparent importance that the classifier is placing on hand orientation.

This series observations make us believe that the transformer is learning to recognize some pattern which is related to hand orientation.

Somewhat more obviously, but also reassuringly, it also seems to perform well for signs sharing hand placement, orientation and movement, and only differing in terms of hand configuration. As mentioned in the introduction, we chose the dataset used in this study based on it’s high variance in terms of hand configuration, even though it lacked in variance for the other components. Therefore we can easily observe many cases that only differ in this regard. See entries in Figure 44 relating to: “A” vs “B”; “LL” vs “RR”; “C” vs “O”; “6” vs “8”; and many others for cases that are identical in each of the 6 components except for hand configuration.

Additionally, it seems that some of the confusions can be explained by similarities in hand configuration. See for example in Figure 44 the cases of confusion for:

* The previously mentioned (“L”, “LL”) and (“R”, “RR”) cases
* “U” and “V”, which only have a slight difference in hand configuration
* “2” and “RR”, which differ in placement, but share hand orientation as well as a similar hand configuration
* “A” and “L”, which are identical except for the index finger being retracted for “A” and extended for “L”
* “1” and “2”, which are identical except for the middle finger being retracted for “1” and extended for “2”
* “M” and “N”, which are identical except for the ring finger being retracted for “N” and extended for “M”

Thus, we believe that, similarly to hand orientation, the transformer is learning to recognize some pattern which is related to hand configuration. This makes us think that the model is, on its own, picking up on parts of the six linguistic components [10] we were trying to isolate. The fact that results got worse when we applied geometric transformations suggests that maybe those steps removed useful information, or at least that they are having some negative impact on the classification task.

One way to test whether our geometric transformations are actually helping or hurting model performance would be to compare them against a wide range of randomized alternatives. We could generate multiple versions of the dataset by applying random transformations without any specific intent to isolate linguistic components. Then, by training models on each of these datasets and recording their performance, we could build a distribution of accuracy scores across these random baselines.

By placing the performance of our handcrafted transformations, as well as the raw untransformed data, within this distribution, we could evaluate whether our approach offers any meaningful advantage. If our transformations perform similarly to random ones—or worse—that would suggest the model does better when left to extract relevant structure from the original landmark space. On the other hand, if our transformations consistently outperform random ones, it would indicate that the overall idea is sound, and that perhaps only specific design choices need to be revised.

Nevertheless, coming back to the results we already obtained, perhaps the connection between the components is the important information that is lost through our forced transformations, as we were trying to completely separate the components. If that is the case, it appears the transformer might be capitalizing on that connection found only in the original, untransformed landmarks, resulting in the improved results.

Another possibility is that our transformations are not properly representing the components, or that our models are not able to interpret the representations we obtain.

For example, we represent hand orientation through the 3 vectors which we chose as the base of our “hand-centric frame of reference”, and it could be that the choice of frame of reference was incorrect. We decided to focus on the plane formed by the palm, and this is in line with the description of hand orientations of Cruz Aldrete et al. [10], but the choice of vectors was somewhat arbitrary and based mostly on intuitive conceptions of hand anatomy.

Additionally, it could be that having part of the data represent coordinates for vectors, which live in one mathematical space, and another part of the data correspond to landmarks, which live in another mathematical space, was hard to reconcile for our models.

Similarly, we tried to represent hand placement through the coordinates of the average point of the hand in a frame of reference extracted from the pose coordinates. Here again, choices for the definition of the frame of reference were somewhat arbitrary, and could lead to a loss of information. But additionally, we also have coordinates for landmarks living in two separate frames of reference.

Up to this point in our analysis, it was unclear to the authors of this work what the effect of clashing these three different mathematical spaces could be for the ability of the transformer to extract relations of the data living in them. However, as we will see in the Comparison between KNN and BERT-mini section, the effect of PH2 on KNN models—an architecture that is simpler and more transparent to study—will give us some insights on this clashing of mathematical spaces.

A separate issue which the confusion matrix points to can be seen by studying the superdiagonal and subdiagonal in Figure 44. It becomes apparent that there is a consistent issue with signs which are contiguous in either alphabetical or numbering order. It is true that there is a relation between closeness in these orders and sign articulation (see for example: the first 5 numbers; “M” and “N”; “S” and “T”). However, knowing that this is also the order in which they were articulated during dataset creation, and that this phenomenon occurs for some more visually distinct but consecutive signs (see for example: “B” and “C”; “F” and “G”; “T” and “U”), we believe that this could also point to an issue in “sign articulation” leakage between videos.

## Live-Inference System Reliability and Limitations

The primary goal of this project is to develop a high-performing, live-inference MSL recognition system for alphanumeric signs. The model we obtained with our methodology obtained excellent results with our testing data, as we can see in Table 12. However, this model performs more poorly as the backbone of our live-inference system where it faces data in a setting that is different from the dataset on which it was trained and tested.

In our exploratory use of the live-inference system based on this “best model”, we observe a few cases:

* For some of the signs, the class is predicted correctly throughout most if not all of the articulation. We see this for: “A”, “D”, “F”, “H”, “K”, “V”, “W”, “2”, “3”, “8”, “10”
* For some signs, the class predicted is that of a sign which shares obvious elements in common with the true class, though overall the similarity varies. We see this for: “C” predicted as “10”, “G” predicted as “H”, “L” predicted as “K”, “LL” predicted as “K”, “P” predicted as “K”, “Q” predicted as “A”, “R” predicted as “V”, “S” predicted as “1”, “U” predicted as “V”, “X” predicted as “H”, “Y” predicted as “9”, “Z” predicted as “K”, “4” predicted as “3”, “5” predicted as “10”, “6” predicted as “G”, “8” predicted as “10” or “H”, “C” predicted as “10”, “Q” predicted as “A” or “6”, “9” predicted as “3”
* For some signs, the prediction is completely off. We see this for “B” predicted as “3”, “E” predicted as “D”, “M” predicted as “10”, “N” predicted as “6”, “O” predicted as “10”, “RR” predicted as “10”, “T” predicted as “P”

Overall, the model is too sensitive to variations, and seems to have a few signs on which it “leans on” when it is unsure (namely: “10”, “V”). These findings suggest overfitting, and reframe our previous conclusions. We can no longer qualify our system as “highly performant”, as it does not act in this fashion for its intended usecase: live inference.

As to why we might be obtaining deceptively excellent results during testing, we believe there might be a few reasons. Because our model is built on top of the MediaPipe landmarking framework—which is itself a very robust system that can deal very well with variations in lighting, camera angles, color palettes and so on [27]—we naively expected to be immune to the issue of overfitting. However, we did not consider that we might end up overfitting on features of the data obtained later on in the data preparation process. In particular, we believe we may be overfitting onto the signers anatomies or signing styles. To address this issue, or to at least obtain more representative scoring metrics, we should in future works use a train/test split that ensures signer separation.

A second possible reason as to why we might be getting excellent performance during testing but not during live inference is that our testing results where those of single experimental runs per training configuration. In the future, work should include statistical significance testing and confidence intervals through multiple experimental repetitions.

Having said this, after discovering that our most accurate testing model was not as performant as we expected it to be for live inference, we decided to try out other models we had trained. Interestingly, we found that the model trained on per-video data, with the same no PH2 and no PH3 data preparation processes, performed much better. In our exploration we found that the sign predictions fell into three cases:

* For some of the signs, and a considerably larger portion than for the per-frame model, the predicted class matches the true class throughout most if not all the articulation. These are: “A”, “C”, “D”, “E”, “F”, “I”, “K”, “L”, “LL”, “M”, “N”, “R”, “RR”, “1”, “2”, “3”, “4”, “6”, “9”, and “10”
* For “J”, we see an interesting phenomenom where for a few instants that we would consider are “the peak of articulation”, but not the full articulation, the sign is correctly predicted, and on either side of those instants, another incorrect but visually similar sign is predicted
* For the rest of the signs, we observe incorrect predictions, but for signs that have some clear element in common with the true class. We see: “B” predicted as “F”, “G” predicted as “8”, “6” or “1”, “H” predicted as “8” or “2”, “O” predicted as “C”, “P” predicted as “3”, “Q”, predicted as “M” or “N”, “S” predicted as “I” or “J”, “T” predicted as “I”, “U” predicted as “R” or “LL”, “V” predicted as “L” or “10”, “W” predicted as “10” or “4”, “Z” predicted as “N”, “5” predicted as “3” or “10”, and “8” predicted as “2”

The model never predicts signs which are seemingly completely different, giving an indication that at least some of the features of all signs are correctly being identified. Additionnally, for the signs which it incorrectly predicts, it does not consistently “fall back” on any sign or group of signs. Overall, this performance indicates no sign of overfitting, and is much more in-line with it’s performance during testing. Due to this, we consider that this model is the one which is most representative of this work.

As to why this system is more robust, we believe it might be that it is less sensitive to variations simply because it is taking data from a prolonged period in time, which might allow it to have a more nuanced representation of the sign being articulated. Another possible explanation is that because the each sample has a much higher dimension count (a multiplicative factor of 12, specifically), it had less of an opportunity to overfit to certain particularities of the training dataset.

An additional challenge justifies its investigation in this exploratory analysis: the case of “J” which we mentioned shows the importance of incorporating “peak of articulation” segmentation models into sign language recognition systems. Doing so should allow for greater accuracy, especially in the case of continuous sign language recognition for which so many moments are transitions between signs.

It is also worth noting that the best KNN model, which functioned on a per-frame basis and performed surprisingly well during testing, suffered of severe overfitting during live-inference, predicting but a handful of signs regardless of the articulation. Since KNN does not involve iterative training, this failure cannot be attributed to typical training dynamics. Instead, it suggests that the per-frame data itself contains strong superficial patterns or biases that KNN can exploit to achieve deceptively high test accuracy. These might be patterns related to the individual signers, unusual pose details, or timing quirks that happen to be consistent in the train and test set but don’t hold up in real-time use and end up confusing the model.The fact that such a simple model can overfit so strongly reinforces the need for careful dataset design—especially with signer separation—and highlights the limitations of per-frame representations that lack temporal context.

## Comparison between KNN and BERT-mini and Implications on Geometric Transformations

A useful comparison can be made between the best-performing KNN models (Table 9) and the results obtained for BERT-Mini (Table 10 and Table 11). While BERT-mini reached a higher accuracy (97.4% compared to 94.1%), the difference is not large. This suggests that, when the input data is clean and well-structured, the classification task is already quite separable—even for a simple model like KNN. It also reinforces the importance of careful data preparation: with the right input representation, simpler models can perform surprisingly well.

Larger differences appear when the input conditions change. One example is the use of per-video data instead of per-frame input. KNN accuracy dropped by more than 30 percentage points in this case (from 94.1% to 59.9%), while BERT-mini dropped by less than 8 points (from 96.7% to 88.99%). This reflects a basic architectural difference. KNN treats each input as a fixed-length vector and has no way to interpret or model time. When a temporal sequence is encoded by concatenating multiple video frames, the model is exposed to more dimensions but gains no understanding of motion or temporal structure. In contrast, BERT-mini is designed to process sequences. Its attention mechanism and multi-layer structure allow it to identify useful temporal patterns and reduce the influence of irrelevant parts of the input. As the system is extended to include more signs that involve motion, this ability becomes more important. The results suggest that transformer models are better suited for handling temporal complexity than models like KNN.

A similar difference can be seen in how the models respond to PH2 transformations. KNN performance dropped from 94.1% to 82.1%, while BERT-Mini dropped by 3.9 points. PH2 transforms each sample’s landmark coordinates into a new frame of reference, defined using fixed points on the palm. Although this is intended to reduce variation caused by viewpoint or signer posture, it introduces another problem: each sample is described in its own coordinate system. Small differences in palm shape or hand pose can cause shifts in how the landmarks are represented. As a result, two samples representing the same sign may appear quite different in the transformed input space.

This change is especially disruptive for KNN. When it compares raw input vectors using some distance (in our case Euclidean distance), it assumes that all samples are expressed in a consistent mathematical space. When PH2 is applied, that assumption no longer holds. Samples are described in different frames of reference, so their coordinates are no longer directly comparable—even when they represent the same gesture. KNN has no way to detect or correct for this misalignment, since it does not learn any internal representation or transformation. BERT-mini is also affected, but to a smaller extent. Although it receives inputs described in variable coordinate frames, it can learn to adjust during training. This helps reduce the impact of PH2, though it does not eliminate it entirely.

These findings also have important implications for a broader hypothesis explored throughout this project: that signs in MSL can be decomposed into six minor units—hand configuration, place of articulation, hand motion, direction of motion, hand orientation, and non-manual cues—and that separating these components could improve model performance. PH2 was developed specifically to support this idea by isolating several of these components, such as configuration and orientation, into distinct, interpretable spatial representations. Earlier results suggested that this approach might be flawed, as the transformation often reduced performance. At the time, this raised doubts about whether the minor units were truly separable, or whether trying to extract them independently was simply misguided.

However, the current analysis provides a more nuanced explanation. The drop in performance may not imply that the components are inseparable, but rather that the method used to isolate them—namely, re-expressing samples in distinct local frames of reference—has important side effects that interfere with downstream modeling. In particular, the lack of a shared coordinate space across samples disrupts comparison and generalization, especially for models like KNN that rely on global geometric consistency. The fact that even BERT-mini, with its capacity to learn internal representations, also experienced a decline—though smaller—suggests that the transformation introduces real friction.

From this, we can draw a more informed, though less conclusive, interpretation: either the six components cannot be separated cleanly, or this is not the right way to separate them. PH2 may distort the geometry in ways that outweigh any benefit from decomposing the sign into its parts. More fundamentally, it may be that attempting to treat the components as modular is incompatible with how signs are realized in practice, especially in a high-dimensional, continuous spatial context. While the hypothesis remains plausible, these findings show that any attempt to extract and model the minor units must carefully consider the interaction between representation, coordinate systems, and model behavior.

Taken together, these findings highlight a key distinction between the two models. KNN performs well when the data is consistent and low in variation, but its performance drops sharply when exposed to temporal sequences or when gestures are represented in inconsistent coordinate systems. BERT-mini is better equipped to handle both types of variation. Its ability to model time and learn invariant features makes it more robust, and more suitable for scaling the system to larger sign vocabularies that involve movement and signer diversity. This distinction was also reflected in the live inference setting, where KNN proved less stable and more sensitive to noise, further limiting its practical applicability.

# CONCLUSIONS AND FUTURE WORK

## Conclusions

This research successfully developed a real-time MSL recognition system for alphanumeric signs. We built a working live-inference system that runs on consumer-grade hardware and is capable of interpreting signs in real time, based on a model which obtained 89.0% accuracy during testing, making it a practical foundation for future tranlation or educational tools. Separately, we trained a model that achieved a test accuracy of 97.4% on isolated, pre-segmented sign data. This high score reflects strong performance under controlled conditions, though it does not fully translate to the challenges and variability encountered during real-time inference with unsegmented, in-the-wild data.

This work used a single, lightweight BERT-mini encoder to recognize both static and dynamic signs within a unified model architecture and training process. While other systems in the broader literature have addressed both sign types jointly, our contribution builds on prior work using this same dataset, where static and dynamic signs were handled separately. Unifying the two into one system simplified the pipeline and will make it easier to deploy and scale.

To efficiently explore the wide space of possible data representations and model configurations, we implemented a two-stage pruning strategy that filtered 546 candidate combinations down to a small set of promising ones. This strategy significantly reduced the number of training runs required, saving both time and computational resources while still guiding us toward the most effective setup.

In our analysis, we found that the highest test accuracy was obtained by training directly on raw hand and pose landmarks, without applying the geometric transformations introduced in PH2. These transformations were designed to isolate linguistic components of signs—such as hand orientation and hand configuration—based on the assumption that signs can be decomposed into six core components. Initially, their underperformance raised questions about the separability of those components. However, further analysis suggests that the performance drop may not result from the hypothesis itself, but from the method used to isolate the components. In particular, PH2 expresses each sample in its own local frame of reference, introducing inconsistencies across the dataset that disrupt generalization, especially for models that rely on global geometric structure. Even in transformer models, which are more robust to such inconsistencies, this approach was found to reduce performance.

At the same time, the model trained on raw data not only achieved higher accuracy but also showed signs of implicitly learning key linguistic features on its own. Confusion matrix analysis revealed that it was particularly sensitive to subtle differences in hand orientation and finger configuration, even without being given any explicit structural decomposition. This suggests that when provided with raw, consistent data, the model was able to discover useful linguistic patterns through training, without requiring handcrafted separation of components.

These results challenge the assumption that separating linguistic features through geometric transformation will necessarily improve recognition. Instead, they suggest that either the components are not easily separable, or that the approach used to separate them needs to be reconsidered. The transformer model appears capable of learning useful feature interactions directly from the full, untransformed data—preserving relationships that handcrafted preprocessing may unintentionally distort or discard.

## Future work

The ultimate goal toward which this project hopes to contribute is the development of a fully functional educational or translation tool for MSL. Achieving this would require more than just accurate sign classification. A complete system would need to include, for example, modules for continuous video processing, automatic sign segmentation, natural language generation (to translate signs into coherent spoken or written sentences), user adaptation (to personalize to different signers), and feedback mechanisms for language learners. Each of these components introduces new research and engineering challenges.

Before reaching that level of system integration, a number of key modeling challenges remain. The current system must first become more robust in real-time conditions. While the live inference pipeline developed here functions end-to-end on consumer-grade hardware, its accuracy lags behind that of the offline testing system, particularly when facing unsegmented input and signing variation. Improving real-time accuracy and stability is an immediate priority. While MediaPipe offers strong robustness to factors like lighting and background variation, the current system still shows performance gaps in live inference. This suggests the need to improve generalization across signers and conditions, possibly by increasing the diversity of the training data.

The first step toward these broader goals is to revisit the data representations and modeling strategies used in this work. One important direction is to refine the geometric transformations introduced in PH2. These transformations were designed to isolate the six linguistic components of signs—such as hand configuration and orientation—by re-expressing each sample in its own local coordinate system. However, our results show that this approach introduces inconsistency across the dataset, disrupting generalization.

We mentioned in the Discussion section a plan to evaluate PH2 against random baselines, but before that, future work should focus on identifying ways to improve on PH2 to isolate linguistic structure while preserving global geometric alignment. A method to accomplish this is unclear to the authors. For instance, adapting the frame of reference determination to some per-video methodology would alleviate frame-to-frame inconsistencies, but cross-video inconsistencies would persist. Additionally, a representation of hand configuration separated from the other components seems to have an intrinsic need for per-frame, hand-centric isolation. Similarly for hand placement, which is by nature an aspect relative to the body, and defined for a moment in time—not a prolonged period. The same can be said for hand orientation, as well.

Perhaps displacing the separation of the components from the data preparation phase of our development, to a point later on, could prove more fruitful with less negative side effects. For example, exploring a system containing a series of models each of which is responsible for predicting a class corresponding to each of the six components, and then having a model built on top of those which incorporates their component predictions to make a final sign prediction, could be a more fruitful investigation.

Whether or not we continue using geometric transformations of some form or another, the model must be extended to support a broader vocabulary of signs. To scale the system beyond isolated alphanumeric signs, two major extensions are required: support for bimanual signsand recognition of continuous signing. Bimanual signs are common in natural MSL and involve coordinated motion between both hands. Supporting them will require the model to track both hands simultaneously, represent their spatial relationship, and capture inter-hand dynamics. This may involve extracting new features—such as relative distances, mirrored motion patterns, or symmetric gestures—and developing spatial encodings that jointly model both hand trajectories within a unified representation. In parallel, the system must be adapted to recognize signs in continuous video streams, where gestures appear without pre-defined segmentation. This setting introduces challenges such as detecting sign boundaries, handling transitions between gestures, and managing variable signing pace and coarticulation effects. Recognizing continuous input will require architectural changes, including mechanisms for temporal segmentation and memory across longer sequences. Given the performance differences observed between KNN and BERT-Mini on video-based input, Transformer models appear especially promising for this task. Future work could explore models that explicitly handle boundary detection and multi-sign context modeling. These extensions demand both new data and changes in the model architecture.

Somewhat independently, a particularly novel direction is to rethink how landmark sequences are encoded into Transformer-based models. Instead of relying only on temporal ordering, future research could explore non-time-based positional encodings, such as those proposed for tree-structured data. For instance, Shiv and Quirk’s [67] method could be adapted to hand landmarks by defining a tree rooted at the wrist, with branches leading through the palm and fingers. This would allow each landmark to serve as a token, and its structural position could be encoded using tree-based positional vectors. Such an approach might help the model capture the anatomical and spatial structure of the hand more directly, rather than relying on flattened frame-wise inputs.

For all future research following this study, more rigorous statistical testing would benefit our analysis. In particular, ensuring signer independence of the training and testing datasets should be incorporated to avoid overfitting. Perhaps using a k-folds cross-validation testing methodology with respect to the signers could thoroughly address this issue. Furthermore, exploring the use of models in a live-inference at different points of their training could help to avoid overfitting. Additionally, statistical significance testing and confidence intervals through multiple experimental repetitions should be included in future works. Finally, cross-dataset testing of our models on international benchmark datasets could allow us to detect overfitting, as well as serving to better place our future works in the larger sign language recognition landscape.

# Bibliography

|  |  |
| --- | --- |
| [1] | C. Valli, C. Lucas, K. Mulrooney and M. Villanueva, Linguistics of American Sign Language: An Introduction, Washington, DC, USA: Gallaudet Univ. Press, 2011. |
| [2] | SIL MÉXICO, "Lengua de Señas en México," [Online]. Available: https://mexico.sil.org/es/lengua\_cultura/lenguas-de-señas-en-méxico. [Accessed 28 Nov 2024]. |
| [3] | G. d. M. Secretaría de Salud, "530. Con discapacidad auditiva, 2.3 millones de personas: Instituto Nacional de Rehabilitación," 28 Nov 2021. [Online]. Available: https://www.gob.mx/salud/prensa/530-con-discapacidad-auditiva-2-3-millones-de-personas-instituto-nacional-de-rehabilitacion?idiom=es#:~:text=En%20México%2C%20aproximadamente%202.3%20millones,ciento%20son%20niñas%20y%20niños.. |
| [4] | INEGI, "Población," 2020. [Online]. Available: https://www.inegi.org.mx/temas/estructura/. |
| [5] | N. Mohamed, M. B. Mustafa and N. Jomhari, "A Review of the Hand Gesture Recognition System: Current Progress and Future Directions," *IEEE Access,* vol. 9, pp. 157422-157436, 2021. |
| [6] | I. A. Adeyanju, O. O. Bello and M. A. Adegboye, "Machine learning methods for sign language recognition: A critical review and analysis," *Intelligent Systems with Application,* vol. 12, 2021. |
| [7] | V. Martínez-Sánchez, I. Villalón-Turrubiates, F. Cervantes-Álvarez and C. Hernández-Mejía, "Exploring a Novel Mexican Sign Language Lexicon Video Dataset," *Multimodal Technologies and Interaction,* 2023. |
| [8] | J. Forster, C. Schmidt, O. Koller, M. Bellgardt and H. Ney, "Extensions of the Sign Language Recognition and Translation Corpus RWTH-PHOENIX-Weather," in *International Conference Language Resources and Evaluation*, Reykjavik, Iceland, 2014. |
| [9] | J. R. González-Rodríguez, D. M. Córdova-Esparza, J. Terven and J. A. Romero-González, "Towards a Bidrectional Mexican Sign Language-Spanish Translation System: A Deep Learning Approach," *Technologies,* vol. 12, no. 1, 2024. |
| [10] | M. Cruz Aldrete, "Gramática de la lengua de señas mexicana," Oct 2008. [Online]. Available: http://elies.rediris.es/elies28/. [Accessed 28 Nov 2024]. |
| [11] | M. E. Rodriguez Trejo, O. Oubram and N. Lakouari, "Modelos Computacionales Aplicados a la Lengua de Señas Mexicana," 2023. |
| [12] | K. Mejía-Pérez, D.-M. Córdova-Esparza, J. Terven, A.-M. Herrera-Navarro, T. García-Ramírez and A. Ramírez-Pedraza, "Automatic Recognition of Mexican Sign language Using a Depth Camera and Recurrent Neural Networks," *Applied Sciences,* 2022. |
| [13] | G. García-Bautista, F. Trujillo-Romero and S. O. Caballero-Morales, "Meixcan sign language recognition using kinect and data time warping algorithm," in *International Conference on Electronics, Communications and Computers (CONIELECOMP)*, Cholula, Mexico, 2017. |
| [14] | C. p. P. y. E. l. D. d. l. CDMX, "Lenguas de señas son fundamentales para el desarrollo de las personas sordas y el acceso a sus derechos," 23 September 2019. [Online]. Available: https://copred.cdmx.gob.mx/comunicacion/nota/lenguas-de-senas-son-fundamentales-para-el-desarrollo-de-las-personas-sordas-y-el-acceso-sus-derechos. [Accessed 20 June 2025]. |
| [15] | IncluSor (Centro para la Inclúsion Social del Sordo A.C.), "Materiales," [Online]. Available: https://www.inclusor.org.mx/index.php/datos-y-descargas/materiales. [Accessed 27 June 2025]. |
| [16] | B. Mocialov, G. Turner, K. Lohan and H. Hastie, "Towards Continuous Sign Language Recognition with Deep Learning," in *Workshop Creating Meaning Robot Assistants*, 2017. |
| [17] | T. Starner and A. Pentland, "Real-time American Sign Language Recognition from video using Hidden Markov Models," in *International Symposium on Computer Vision*, Coral Gables, FL, USA, 1995. |
| [18] | S. C. Dimri, R. Indu, H. S. Negi, N. Panwar and M. Sarda, "Hidden Markov Model - Applications, Strengths, and Weaknesses," in *International Conference on Device Intelligence, Computing and Communication Technologies*, Dehradun, India, 2024. |
| [19] | U. Farooq, M. Rahim, N. Sabir, A. Hussain et al., "Advances in machine translation for sign language: approaches, limitations, and challenges," *Neural Computing and Applications,* 2021. |
| [20] | C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning,* vol. 20, no. 3, pp. 273-297, 1995. |
| [21] | F. Chollet, Deep Learning with Python, Second Edition, Manning Publications, 2021. |
| [22] | Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998. |
| [23] | K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," 2 Dec 2015. [Online]. Available: https://arxiv.org/abs/1511.08458. [Accessed 1 Dec 2024]. |
| [24] | O. Koller, S. Zargaran, H. Ney and R. Bowden, "Deep Sign: Enabling Robust Statistical Continuous Sign Language Recognition via Hybrid CNN-HMMs". |
| [25] | J. Huang, W. Zhou, H. Li and W. Li, "Sign Language Recognition using 3D convolutional neural networks," in *IEEE International Conference on Multimedia and Expo (ICME)*, Turin, 2015. |
| [26] | C. P. C. Lab, "OpenPose: Main Page," [Online]. Available: https://cmu-perceptual-computing-lab.github.io/openpose/web/html/doc/index.html. [Accessed 1 Dec 2024]. |
| [27] | F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C.-L. Chang and M. Grundmann, "MediaPipe Hands: On-device Real-time Hand Tracking," 2020. [Online]. Available: https://arxiv.org/abs/2006.10214. |
| [28] | Google, "Hand landmarks detection guide," Google, 21 May 2024. [Online]. Available: https://ai.google.dev/edge/mediapipe/solutions/guide. [Accessed 28 Nov 2024]. |
| [29] | J. L. Elman, "Finding Structure in time," *Cognitive Science,* vol. 14, no. 2, pp. 179-211, 1990. |
| [30] | S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation,* vol. 9, no. 8, pp. 1735-1780, 1997. |
| [31] | K. Cho et al., "Learning phrase representations using RNN encoder-decoder fir statistical machine translation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014. |
| [32] | A. Vaswani et al., "Attention is all you need," in *Advances in Neiral Information Processing Systems (NeurIPS)*, 2017. |
| [33] | F. A. Mendoza López, J. Rodriguez and F. Martínez, "A multitask transformer to sign language translation using motion gesture primitives," 25 March 2025. [Online]. Available: https://arxiv.org/abs/2503.19668. [Accessed 27 June 2025]. |
| [34] | N. C. Camgoz, S. Hadfield, O. Koller and R. Bowden, "Sign Language Transformers: Joint end-to-end sign language recognition and translation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. |
| [35] | X. Zhang, Y. Wang, M. Zuo and M. Lin, "Contextual sequence transduction for sign language recognition with language modeling," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. |
| [36] | Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *NAACL,* 2019. |
| [37] | I. Turc, M.-W. Change, K. Lee and K. Toutanova, "Well-Read Students Learn Better: On the Importance of Pre-training Compact Models," 25 September 2019. [Online]. Available: https://arxiv.org/abs/1908.08962. [Accessed 22 June 2025]. |
| [38] | V. Sanh, L. Debut, J. Chaumond and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," 1 March 2020. [Online]. Available: https://arxiv.org/abs/1910.01108. [Accessed 22 June 2025]. |
| [39] | B. Y. Marquez, A. Alanis, A. Quezada and J. S. Magdaleno-Palencia, "Development of a Mobile Application with Artificial Intelligence for Mexican Sign Language Recognition," *International Journal of Interactive Mobile Technologies,* vol. 19, no. 9, pp. 122-139, 2025. |
| [40] | K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 10 April 2015. [Online]. Available: https://arxiv.org/abs/1409.1556. [Accessed 27 June 2025]. |
| [41] | H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Trans. Accoust., Speech, Signal Process.,* vol. 26, no. 1, pp. 43-49, 1978. |
| [42] | E. J. M. Cataño, J. C. Estrada Gutiérrez, A. B. Flores Jiménez, M. Cano Lara and M. Jiménez Rodríguez, "Detección del abecedario de Lengua de Señas Mexicanas (LSM) usando MediaPipe, SVM y Random Forest," 2024. |
| [43] | L. Breiman, "Random forests," *Machine Learning,* vol. 45, no. 1, pp. 5-32, 2001. |
| [44] | M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.,* vol. 45, no. 11, pp. 2673-2681, 1997. |
| [45] | M. Rodriguez, O. Oubram, A. Bassam, N. Lakouari and R. Tariq, "Mexican Sign Language recognition: Dataset creation and performance evaluation using MediaPipe and machine learning techniques," *Electronics,* vol. 14, no. 7, 2025. |
| [46] | M. E. Rodriguez-Trejo, O. Oubram, B. ALI and N. Lakouari, "Mexican Sign Language's Dactylology and Ten First Numbers - Labeled images and videos. From person #1 to #5," 30 May 2023. [Online]. Available: https://data.mendeley.com/datasets/5s4mt7xrd9/1. [Accessed 1 Dec 2024]. |
| [47] | M. E. Rodriguez-Trejo, O. Oubram, B. ALI and N. Lakouari, "Mexican Sign Language's Dactylology and Ten First Numbers - Labeled images and videos. From person #6 to #11," 30 May 2023. [Online]. Available: https://data.mendeley.com/datasets/67htnzmwbb/1. [Accessed 1 Dec 2024]. |
| [48] | K. Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, 2012. |
| [49] | J. Jiao, X. Sun, M. Chen and H. Wang, "CoSign: Pose-based continuous sign language recognition via two-stream graph convolution," in *IEEE/CVF Conference on Computer Vision (ICCV)*, 2023. |
| [50] | M. Zuo, Y. Wang, M. Lin and X. Zhang, "Natural language-assisted sign language recognition," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. |
| [51] | H. Shi, H. Xu and K. Zhang, "Multimodel sign language recognition using vision transformers," in *IEEE International Conference on Multimedia and Expo (ICME)*, 2023. |
| [52] | X. Zhou, Y. Li, J. Xu and B. Yang, "Hierarchical vision transformer for isolated sign language recognition," *Pattern Recognition Letters,* vol. 169, pp. 1-9, 2023. |
| [53] | I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, Cambridge, MA, USA: MIT Press, 2016. |
| [54] | I. T. Jolliffe, Principal Component Analysis, New York, NY, USA: Springer, 2002. |
| [55] | McInnes et al., "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction," *Journal of Open Source Software,* 2018. |
| [56] | K. Cho, B. van Merrienboer, D. Bahdanau and Y. Bengio, "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches," 7 October 2014. [Online]. Available: https://arxiv.org/abs/1409.1259. [Accessed 22 June 2025]. |
| [57] | C. D. Manning, P. Raghavan and H. Schütze, Introduction to Information Retrieval, Cambridge University Press, 2008. |
| [58] | OpenCV, "OpenCV - Open Computer Vision Library," [Online]. Available: https://opencv.org. [Accessed 28 Nov 2024]. |
| [59] | T. Tran, "gaunerst/bert-tiny-uncased," 2024 October 2024. [Online]. Available: https://huggingface.co/gaunernst/bert-tiny-uncased. [Accessed 22 June 2025]. |
| [60] | T. Tran, "gaunernst/bert-mini-uncased," 25 October 2024. [Online]. Available: https://huggingface.co/gaunernst/bert-mini-uncased. [Accessed 22 June 2025]. |
| [61] | T. Tran, "gaunernst/bert-small-uncased," 4 January 2025. [Online]. Available: https://huggingface.co/gaunernst/bert-small-uncased. [Accessed 22 June 2025]. |
| [62] | T. Tran, "gaunernst/bert-medium-uncased," 04 January 2025. [Online]. Available: https://huggingface.co/gaunernst/bert-medium-uncased. [Accessed 22 June 2025]. |
| [63] | H. F. t. team, "distilbert/distilbert-base-uncased," [Online]. Available: https://huggingface.co/distilbert/distilbert-base-uncased. [Accessed 22 June 2025]. |
| [64] | J. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," *Berkeley Symposium on Mathematical Statistics and Probabilty,* vol. 1, pp. 281-297, 1967. |
| [65] | PyTorch, "AdamW," [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.optim.AdamW.html. [Accessed 23 June 2025]. |
| [66] | PyTorch, "CrossEntropyLoss," [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html. [Accessed 23 June 2025]. |
| [67] | V. Shiv and C. Quirk, "Novel Positional Encodings to Enable Tree-Based Transformers," *Advances in Neural Information Processing Systems,* 2019. |
| [68] | O. Koller, N. C. Camgoz, H. Ney and R. Bowden, "Weakly supervised learning with multi-stream CNN-LSTM-HMMs to discover sequential parallelism in sign language videos," *IEEE Trans. Pattern Anal. Mach. Intell.,* vol. 42, no. 9, pp. 2306-2320, 2020. |
| [69] | Z. Cui, X. Liu and H. Zhang, "Deep learning-based sign language recognition using convolutional neural networks and LSTM," *Mutlimedia Tools Appl.,* vol. 78, no. 11, pp. 15137-15332, 2019. |
| [70] | N. C. Camgoz, O. Koller, S. Hadfield and R. Bowden, "Sign language transformers: Joint end-to-end sign language recognition and translation," in *IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, 2020. |
| [71] | Scikit-Learn, "PCA," [Online]. Available: https://scikit-learn.org/dev/modules/generated/sklearn.decomposition.PCA.html. [Accessed 28 Nov 2024]. |
| [72] | Scikit-Learn, "KNeighborsClassifier," [Online]. Available: https://scikit-learn.org/dev/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. [Accessed 28 Nov 2024]. |
| [73] | T. Tran, "google-bert/bert-base-uncased," Google, [Online]. Available: https://huggingface.co/google-bert/bert-base-uncased. [Accessed 22 June 2025]. |
| [74] | OpenAI, "ChatGPT," OpenAI, [Online]. Available: https://chat.openai.com/. [Accessed 24 June 2025]. |
| [75] | D. Santa Cruz, "TOG," [Online]. Available: https://github.com/dasantac/TOG/tree/dev4. [Accessed 30 Jun 2025]. |

**APPENDIX A. Geometric Transformations**

The following pseudocode summarizes the geometric transformation process described in Section 4.3.3. We start with a series of 5 helper functions, and end with the main driving process.

Function compute\_hand\_centroid(hand\_landmarks):

x\_list = all x values from the 21 hand landmarks

y\_list = all y values from the 21 hand landmarks

z\_list = all z values from the 21 hand landmarks

mean\_x = average of x\_list

mean\_y = average of y\_list

mean\_z = average of z\_list

return [mean\_x, mean\_y, mean\_z]

Function compute\_hand\_frame(hand\_landmarks, handedness):

p0 = wrist

pA = average of points at base of index and middle fingers

pB = average of points at base of ring and pinky fingers

v1 = pA - p0

v2 = pB - p0

v3 = cross product of v1 and v2

if handedness is "Right":

flip direction of v3

Normalize v1, v2, and v3

return [v1, v2, v3]

Function project\_hand\_landmarks(hand\_landmarks, wrist, hand\_frame):

# Step 1: Move all points so that the wrist is the origin

For each point in the hand:

subtract wrist coordinates from the point

# Step 2: Build a matrix using the hand frame vectors

B = matrix with rows [v1, v2, v3]

I = inverse of B

# Step 3: Apply the change of basis

For each translated point:

new\_point = I × point

save new\_point

return all transformed hand landmarks

Function compute\_body\_frame(pose\_landmarks):

p0 = nose

pA = left shoulder

pB = right shoulder

v1 = pA - p0

v2 = pB - p0

v3 = cross product of v1 and v2

Normalize v1, v2, and v3

return [v1, v2, v3]

Function project\_hand\_centroid(centroid, nose, body\_frame):

# Step 1: Move centroid so nose is at origin

shifted = centroid - nose

# Step 2: Invert the body frame matrix

I = inverse of matrix [v1, v2, v3]

# Step 3: Apply transformation

transformed = I × shifted

return transformed

Function apply\_PH2\_transformations(dataframe):

For each row in the original dataframe:

# Step 1: Compute hand centroid

h\_mean = compute\_hand\_centroid(hand\_landmarks)

# Step 2: Compute hand orientation frame

h\_frame = compute\_hand\_frame(hand\_landmarks, handedness)

# Step 3: Project hand landmarks into hand frame

transformed\_hand = project\_hand\_landmarks(hand\_landmarks, wrist, h\_frame)

# Step 4: Compute body orientation frame

body\_frame = compute\_body\_frame(pose\_landmarks)

# Step 5: Project hand centroid into body frame

transformed\_centroid = project\_hand\_centroid(h\_mean, nose, body\_frame)

# Step 6: Store transformed values in new columns

Return a new dataframe with the transformed columns

**APPENDIX B. Dimensionality Reduction Visualizations**

In this appendix we go over the 3-dimensional visualizations obtained using PCA, PCA combined with a kernel, and UMAP.

The graphics are organized by the following set of rules:

* First, by whether or not the source data which was reduced was or was not geometrically transformed (PH2) prior to the reduction (PH3).
* Second, by data unit. Here we mean whether:
  + It includes the data for both active and inactive hands, and the label column is active\_hand. This will be seen in the title of the figure as AHpf
  + It includes only the data for active hands, and each row corresponds to a frame. This will be seen in the title of the figure as Spf
  + It includes only the data for active hands, and each row corresponds to a video. This will be seen in the title of the figure as Spv
* Third, by the reducer being used (PCA, PCA with a kernel, UMAP).
* Finally, if the reducer used a kernel, by the kernel in question (cosine, polynomial, RBF, or sigmoid).

|  |  |
| --- | --- |
| Figure 47: No PH2, active hand detection, PCA, no kernel | Figure 48: No PH2, active hand detection, PCA, cosine kernel |
| Figure 49: No PH2, active hand detection, PCA, polynomial kernel | Figure 50: No PH2, active hand detection, PCA, RBF kernel |
| Figure 51: No PH2, active hand detection, PCA, sigmoid kernel | Figure 52: No PH2, active hand detection, UMAP |
| Figure 53: No PH2, per-frame sign recognition, PCA, no kernel | Figure 54: No PH2, per-frame sign recognition, PCA, cosine kernel |
| Figure 55: No PH2, per-frame sign recognition, PCA, polynomial kernel | Figure 56: No PH2, per-frame sign recognition, PCA, RBF kernel |
| Figure 57: No PH2, per-frame sign recognition, PCA, sigmoid kernel | Figure 58: No PH2, per-frame sign recognition, UMAP |
| Figure 59: No PH2, per-video sign recognition, PCA, no kernel | Figure 60: No PH2, per-video sign recognition, PCA, cosine kernel |
| Figure 61: No PH2, per-video sign recognition, PCA, polynomial kernel | Figure 62: No PH2, per-video sign recognition, PCA, RBF kernel |
| Figure 63: No PH2, per-video sign recognition, PCA, sigmoid kernel | Figure 64: No PH2, per-video sign recognition, UMAP |
| Figure 65: PH2 transformed, active hand detection, PCA, no kernel | Figure 66: PH2 transformed, active hand detection, PCA, cosine kernel |
| Figure 67: PH2 transformed, active hand detection, PCA, polynomial kernel | Figure 68: PH2 transformed, active hand detection, PCA, RBF kernel |
| Figure 69: PH2 transformed, active hand detection, PCA, sigmoid kernel | Figure 70: PH2 transformed, active hand detection, UMAP |
| Figure 71: PH2 transformed, per-frame sign recognition, PCA, no kernel | Figure 72: PH2 transformed, per-frame sign recognition, PCA, cosine kernel |
| Figure 73: PH2 transformed, per-frame sign recognition, PCA, polynomial kernel | Figure 74: PH2 transformed, per-frame sign recognition, PCA, RBF kernel |
| Figure 75: PH2 transformed, per-frame sign recognition, PCA, sigmoid kernel | Figure 76: PH2 transformed, per-frame sign recognition, UMAP |
| Figure 77: PH2 transformed, per-video sign recognition, PCA, no kernel | Figure 78: PH2 transformed, per-video sign recognition, PCA, cosine kernel |
| Figure 79: PH2 transformed, per-video sign recognition, PCA, polynomial kernel | Figure 80: PH2 transformed, per-video sign recognition, PCA, RBF kernel |
| Figure 81: PH2 transformed, per-video sign recognition, PCA, sigmoid kernel | Figure 82: PH2 transformed, per-video sign recognition, UMAP |

**APPENDIX C. Confusion Matrices and Training Loss Evolution for the Final Candidate BERT-mini Models**

|  |
| --- |
| Figure 83: Confusion matrix for BERT-mini model trained on per-frame data, with no PH2, with no PH3 |
| Figure 84: Training loss evolution for BERT-mini model trained on per-frame data, with no PH2, with no PH3 |

In this appendix we show training and testing visualizations that compare how BERT-mini combined with different data formats and preparation processes.

|  |
| --- |
| Figure 85: Confusion matrix for BERT-mini model trained on per-frame data, with no PH2, with PH3 |
| Figure 86: Training loss evolution for BERT-mini model trained on per-frame data, with no PH2, with PH3 |

|  |
| --- |
| Figure 87: Confusion matrix for BERT-mini model trained on per-frame data, with PH2, with no PH3 |
| Figure 88: Training loss evolution for BERT-mini model trained on per-frame data, with PH2, with no PH3 |

|  |
| --- |
| Figure 89: Confusion matrix for BERT-mini model trained on per-frame data, with PH2, with PH3 |
| Figure 90: Training loss evolution for BERT-mini model trained on per-frame data, with PH2, with PH3 |

|  |
| --- |
| Figure 91: Confusion matrix for BERT-mini model trained on per-video data, with no PH2, with no PH3 |
| Figure 92: Training loss evolution for BERT-mini model trained on per-video data, with no PH2, with no PH3 |

|  |
| --- |
| Figure 93: Confusion matrix for BERT-mini model trained on per-video data, with no PH2, with PH3 |
| Figure 94: Training loss evolution for BERT-mini model trained on per-video data, with no PH2, with PH3 |

|  |
| --- |
| Figure 95: Confusion matrix for BERT-mini model trained on per-video data, with PH2, with no PH3 |
| Figure 96: Training loss evolution for BERT-mini model trained on per-video data, with PH2, with no PH3 |

|  |
| --- |
| Figure 97: Confusion matrix for BERT-mini model trained on per-video data, with PH2, with PH3 |
| Figure 98: Training loss evolution for BERT-mini model trained on per-video data, with PH2, with PH3 |

**APPENDIX D. Confusion Matrix for the most accurate KNN Sign Classifier**

|  |
| --- |
| Figure 99: Confusion matrix for KNN model trained on per-frame data, with no PH2, with no PH3 |

1. MediaPipe Hands sometimes, although rarely, fails to detect existing hands in a frame. In our case, out of the 3904 videos in our filtered dataset, on each of which we used the model to detect hands in the 12 middle frames, there were only 3 frames were no hands were detected and there should have been. We decided to drop the videos containing those frames, as their low number did not justify the time investment needed to adapt our process to them. [↑](#footnote-ref-1)