**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 lexical 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, in live-inference conditions it tends to produce the same predictions regardless of input, suggesting overfitting. Other configurations, while scoring lower in testing, respond more to variations in input and behave more plausibly in real-time use.

We conclude with a discussion of trade-offs between accuracy and robustness, and propose future work to understand and improve the geometric transformations.

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 léxica 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 manos y 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 cientos 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, alcanza 97.4 % de exactitud en pruebas. Sin embargo, en inferencia en tiempo real tiende a predecir siempre las mismas señas, sugiriendo un sobreajuste. Otras configuraciones, aunque con menor desempeño en las pruebas, responden mejor a las variaciones de entrada y se comportan de manera más convincente.

Finalmente, se discuten los compromisos entre precisión y robustez, y se proponen estrategias para entender y mejorar las transformaciones geométricas.

TABLE OF CONTENTS

[ACKNOWLEDGEMENTS 3](#_Toc201972700)

[AGRADECIMIENTOS 4](#_Toc201972701)

[DEDICATION 5](#_Toc201972702)

[DEDICATORIA 6](#_Toc201972703)

[ABSTRACT 7](#_Toc201972704)

[RESUMEN 8](#_Toc201972705)

[TABLE OF CONTENTS 9](#_Toc201972706)

[LIST OF FIGURES 11](#_Toc201972707)

[LIST OF TABLES 14](#_Toc201972708)

[LIST OF ACRONYMS AND ABBREVIATIONS 15](#_Toc201972709)

[1. INTRODUCTION 16](#_Toc201972710)

[1.1. Mexican Sign Language background 18](#_Toc201972711)

[2. STATE OF THE ART 24](#_Toc201972712)

[2.1. Historical progression in Sign Language Recognition 25](#_Toc201972713)

[2.2 Mexican Sign Language Recognition 27](#_Toc201972714)

[3. THEORETICAL FRAMEWORK 30](#_Toc201972715)

[3.1 Lexical structure of Mexican Sign Language 31](#_Toc201972716)

[3.2 Feature Extraction and Representation 31](#_Toc201972717)

[3.2.1 Convolution Neural Networks (CNN) 31](#_Toc201972718)

[3.2.2 Landmark Extraction with MediaPipe 32](#_Toc201972719)

[3.3 Dimensionality reduction of features 32](#_Toc201972720)

[3.4 Neural Network Architectures for Sequential Modeling 32](#_Toc201972721)

[3.5 Evaluation Metrics for Classification Models 33](#_Toc201972722)

[4. METHODOLOGY 35](#_Toc201972723)

[4.1 Methodology overview 36](#_Toc201972724)

[4.2 Raw dataset 37](#_Toc201972725)

[4.2.1 Data organization and filtering 38](#_Toc201972726)

[4.3 Data preparation 39](#_Toc201972727)

[4.3.1 Preparation overview 39](#_Toc201972728)

[4.3.2 (PH1) Video to landmarks 39](#_Toc201972729)

[4.3.2.1 Purpose of using landmarks instead of images 39](#_Toc201972730)

[4.3.2.2 Basic use of MediaPipe 40](#_Toc201972731)

[4.3.2.3 Use of MediaPipe for this project 41](#_Toc201972732)

[4.3.2.4 PH1 output description 43](#_Toc201972733)

[4.3.3 (PH2) Landmark Transformations 43](#_Toc201972734)

[4.3.3.1 PH2 output description 45](#_Toc201972735)

[4.3.4 (PH3) Dimensionality reduction 45](#_Toc201972736)

[4.3.4.1 3-component reduction visualization insights 46](#_Toc201972737)

[4.4 Dataset and model selection 52](#_Toc201972738)

[4.4.1 Goal 52](#_Toc201972739)

[4.4.2 Dataset selection 53](#_Toc201972740)

[4.4.2.1 Selection of subsets using K-Means 54](#_Toc201972741)

[4.4.2.2 Fit K Nearest Neighbors models for each subset, for each dataset candidate 55](#_Toc201972742)

[4.4.2.2.1 Computing significance scores based on observed accuracy 55](#_Toc201972743)

[4.4.2.2.2 Comparing models across number of classes 57](#_Toc201972744)

[4.4.2.2.3 Active hand detection model 62](#_Toc201972745)

[4.4.2.2.4 Best KNN Sign Recognition 63](#_Toc201972746)

[4.4.3 Model selection 64](#_Toc201972747)

[4.4.4 Final selection 67](#_Toc201972748)

[4.5 Best model finetuning 68](#_Toc201972749)

[5. RESULTS AND DISCUSSION 69](#_Toc201972750)

[5.1 Results 70](#_Toc201972751)

[5.2 Discussion 71](#_Toc201972752)

[5.2.1 Comparing to similar studies 71](#_Toc201972753)

[5.2.2 Interpretation of the confusion matrix 73](#_Toc201972754)

[5.2.3 Live-inference system 76](#_Toc201972755)

[5.2.4 Comparison between KNN and BERT-mini 79](#_Toc201972756)

[6. CONCLUSIONS AND FUTURE WORK 81](#_Toc201972757)

[6.1. Conclusions 82](#_Toc201972758)

[6.2. Future work 83](#_Toc201972759)

[7. Bibliography 85](#_Toc201972760)

LIST OF FIGURES

[Figure 1: MSL alphabet, from [15] 19](#_Toc201972761)

[Figure 2: MSL numbers, from [15] 19](#_Toc201972762)

[Figure 3: Hand orientations as shown in [10] 21](#_Toc201972763)

[Figure 4: Hand landmarks extracted by MediaPipe 26](#_Toc201972764)

[Figure 5: Methodology overview 37](#_Toc201972765)

[Figure 6: Data Organization and Filtering 38](#_Toc201972766)

[Figure 7: Data Preparation 39](#_Toc201972767)

[Figure 8: MediaPipe Hand Landmarks 40](#_Toc201972768)

[Figure 9: MediaPipe Pose Landmarks 40](#_Toc201972769)

[Figure 10: Phase 1: Landmark Extraction 41](#_Toc201972770)

[Figure 11: Phase 2: Landmark Transformations 44](#_Toc201972771)

[Figure 12: PH3: Dimensionality reduction 46](#_Toc201972772)

[Figure 13: 3-component reduction of the PH2-processed per-frame data using PCA with a cosine kernel and coloring by sign 47](#_Toc201972773)

[Figure 14: 3-component reduction of the non-PH2-processed per-frame data using PCA with a cosine kernel and coloring by sign 47](#_Toc201972774)

[Figure 15: Using a cosine kernel 48](#_Toc201972775)

[Figure 16: Using a polynomial kernel 48](#_Toc201972776)

[Figure 17: Using a radial basis kernel 48](#_Toc201972777)

[Figure 18: Using a sigmoid kernel 48](#_Toc201972778)

[Figure 19: Using a cosine kernel 49](#_Toc201972779)

[Figure 20: Using a polynomial kernel 49](#_Toc201972780)

[Figure 21: Using a radial basis kernel 49](#_Toc201972781)

[Figure 22: Using a sigmoid kernel 49](#_Toc201972782)

[Figure 23: 3-component reduction of the PH2-processed per-frame data using PCA with a radial-based function kernel and coloring by sign 50](#_Toc201972783)

[Figure 24: 3-component reduction of the non-PH2-processed per-frame data using PCA with a radial-based function kernel and coloring by sign 50](#_Toc201972784)

[Figure 25: 3-component reduction of the PH2-processed per-video data using PCA with a radial-based function kernel and coloring by sign 51](#_Toc201972785)

[Figure 26: 3-component reduction of the non-PH2-processed per-video data using PCA with a radial-based function kernel and coloring by sign 51](#_Toc201972786)

[Figure 27: 3-component reduction of the PH2-processed per-frame data using UMAP and coloring by sign 52](#_Toc201972787)

[Figure 28: 3-component reduction of the PH2-processed per-video data using UMAP and coloring by sign 52](#_Toc201972788)

[Figure 29: Dataset and Model Selection 53](#_Toc201972789)

[Figure 30: Dataset Selection 54](#_Toc201972790)

[Figure 31: Mean of the top 10 significance scores for each number of classes studied 57](#_Toc201972791)

[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 58](#_Toc201972792)

[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) 59](#_Toc201972793)

[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) 60](#_Toc201972794)

[Figure 35: Evolution over the number of classes of the significance scores obtained using different reduction techniques 60](#_Toc201972795)

[Figure 36: Evolution over the number of classes of the significance scores obtained combining different kernels with PCA 61](#_Toc201972796)

[Figure 37: Mean of the top 10 accuracy scores for different number of components in a PCA reducer 62](#_Toc201972797)

[Figure 38: Mean of the top 3 accuracies by loaded model 65](#_Toc201972798)

[Figure 39: Mean of the top 3 accuracies by unit of data. Blue: per-frame; orange: per-video. 65](#_Toc201972799)

[Figure 40: Top 3 Mean accuracy for each boolean pair of use of (PH2, PH3) as part of data preparation, for all model architectures 66](#_Toc201972800)

[Figure 41: Training loss evolution for best model 70](#_Toc201972801)

[Figure 42: Confusion matrix for best model 71](#_Toc201972802)

[Figure 43: MSL alphabet, from [15] 73](#_Toc201972803)

[Figure 44: MSL numbers, from [15] 73](#_Toc201972804)

[Figure 45: Per frame data 77](#_Toc201972805)

[Figure 46: Per video data 77](#_Toc201972806)

[Figure 47: Confusion matrix for the per-frame, PH3-based model used for live-inference 78](#_Toc201972807)

[Figure 48: Confusion matrix for the per-video, PH3-based model used for live-inference 78](#_Toc201972808)

[Figure 49: No PH2, active hand detection, PCA, no kernel 93](#_Toc201972809)

[Figure 50: No PH2, active hand detection, PCA, cosine kernel 93](#_Toc201972810)

[Figure 51: No PH2, active hand detection, PCA, polynomial kernel 94](#_Toc201972811)

[Figure 52: No PH2, active hand detection, PCA, RBF kernel 94](#_Toc201972812)

[Figure 53: No PH2, active hand detection, PCA, sigmoid kernel 94](#_Toc201972813)

[Figure 54: No PH2, active hand detection, UMAP 94](#_Toc201972814)

[Figure 55: No PH2, per-frame sign recognition, PCA, no kernel 95](#_Toc201972815)

[Figure 56: No PH2, per-frame sign recognition, PCA, cosine kernel 95](#_Toc201972816)

[Figure 57: No PH2, per-frame sign recognition, PCA, polynomial kernel 95](#_Toc201972817)

[Figure 58: No PH2, per-frame sign recognition, PCA, RBF kernel 95](#_Toc201972818)

[Figure 59: No PH2, per-frame sign recognition, PCA, sigmoid kernel 96](#_Toc201972819)

[Figure 60: No PH2, per-frame sign recognition, UMAP 96](#_Toc201972820)

[Figure 61: No PH2, per-video sign recognition, PCA, no kernel 96](#_Toc201972821)

[Figure 62: No PH2, per-video sign recognition, PCA, cosine kernel 96](#_Toc201972822)

[Figure 63: No PH2, per-video sign recognition, PCA, polynomial kernel 97](#_Toc201972823)

[Figure 64: No PH2, per-video sign recognition, PCA, RBF kernel 97](#_Toc201972824)

[Figure 65: No PH2, per-video sign recognition, PCA, sigmoid kernel 97](#_Toc201972825)

[Figure 66: No PH2, per-video sign recognition, UMAP 97](#_Toc201972826)

[Figure 67: PH2 transformed, active hand detection, PCA, no kernel 98](#_Toc201972827)

[Figure 68: PH2 transformed, active hand detection, PCA, cosine kernel 98](#_Toc201972828)

[Figure 69: PH2 transformed, active hand detection, PCA, polynomial kernel 98](#_Toc201972829)

[Figure 70: PH2 transformed, active hand detection, PCA, RBF kernel 98](#_Toc201972830)

[Figure 71: PH2 transformed, active hand detection, PCA, sigmoid kernel 99](#_Toc201972831)

[Figure 72: PH2 transformed, active hand detection, UMAP 99](#_Toc201972832)

[Figure 73: PH2 transformed, per-frame sign recognition, PCA, no kernel 99](#_Toc201972833)

[Figure 74: PH2 transformed, per-frame sign recognition, PCA, cosine kernel 99](#_Toc201972834)

[Figure 75: PH2 transformed, per-frame sign recognition, PCA, polynomial kernel 100](#_Toc201972835)

[Figure 76: PH2 transformed, per-frame sign recognition, PCA, RBF kernel 100](#_Toc201972836)

[Figure 77: PH2 transformed, per-frame sign recognition, PCA, sigmoid kernel 100](#_Toc201972837)

[Figure 78: PH2 transformed, per-frame sign recognition, UMAP 100](#_Toc201972838)

[Figure 79: PH2 transformed, per-video sign recognition, PCA, no kernel 101](#_Toc201972839)

[Figure 80: PH2 transformed, per-video sign recognition, PCA, cosine kernel 101](#_Toc201972840)

[Figure 81: PH2 transformed, per-video sign recognition, PCA, polynomial kernel 101](#_Toc201972841)

[Figure 82: PH2 transformed, per-video sign recognition, PCA, RBF kernel 101](#_Toc201972842)

[Figure 83: PH2 transformed, per-video sign recognition, PCA, sigmoid kernel 102](#_Toc201972843)

[Figure 84: PH2 transformed, per-video sign recognition, UMAP 102](#_Toc201972844)

[Figure 85: Confusion matrix for BERT-mini model trained on per-frame data, with no PH2, with no PH3 103](#_Toc201972845)

[Figure 86: Training loss evolution for BERT-mini model trained on per-frame data, with no PH2, with no PH3 103](#_Toc201972846)

[Figure 87: Confusion matrix for BERT-mini model trained on per-frame data, with no PH2, with PH3 104](#_Toc201972847)

[Figure 88: Training loss evolution for BERT-mini model trained on per-frame data, with no PH2, with PH3 104](#_Toc201972848)

[Figure 89: Confusion matrix for BERT-mini model trained on per-frame data, with PH2, with no PH3 105](#_Toc201972849)

[Figure 90: Training loss evolution for BERT-mini model trained on per-frame data, with PH2, with no PH3 105](#_Toc201972850)

[Figure 91: Confusion matrix for BERT-mini model trained on per-frame data, with PH2, with PH3 106](#_Toc201972851)

[Figure 92: Training loss evolution for BERT-mini model trained on per-frame data, with PH2, with PH3 106](#_Toc201972852)

[Figure 93: Confusion matrix for BERT-mini model trained on per-video data, with no PH2, with no PH3 107](#_Toc201972853)

[Figure 94: Training loss evolution for BERT-mini model trained on per-video data, with no PH2, with no PH3 107](#_Toc201972854)

[Figure 95: Confusion matrix for BERT-mini model trained on per-video data, with no PH2, with PH3 108](#_Toc201972855)

[Figure 96: Training loss evolution for BERT-mini model trained on per-video data, with no PH2, with PH3 108](#_Toc201972856)

[Figure 97: Confusion matrix for BERT-mini model trained on per-video data, with PH2, with no PH3 109](#_Toc201972857)

[Figure 98: Training loss evolution for BERT-mini model trained on per-video data, with PH2, with no PH3 109](#_Toc201972858)

[Figure 99: Confusion matrix for BERT-mini model trained on per-video data, with PH2, with PH3 110](#_Toc201972859)

[Figure 100: Training loss evolution for BERT-mini model trained on per-video data, with PH2, with PH3 110](#_Toc201972860)

[Figure 101: Confusion matrix for KNN model trained on per-frame data, with no PH2, with no PH3 111](#_Toc201972861)

LIST OF TABLES

[Table 1: Description of alphanumeric signs by element 22](#_Toc201972862)

[Table 2: State of the Art MSL systems comparison 29](#_Toc201972863)

[Table 3: Samples from the alphanumeric MSL dataset 37](#_Toc201972864)

[Table 4: Comparison of PH2-processed per-frame data using PCA and 4 different kernels 48](#_Toc201972865)

[Table 5: Comparison of non-PH2-processed per-frame data using PCA and 4 different kernels 49](#_Toc201972866)

[Table 6: Size comparison of different BERT inspired architectures as seen in [53] 53](#_Toc201972867)

[Table 7: Active hand detection models best scores 62](#_Toc201972868)

[Table 8: Scores for best KNN alphanumeric sign classifiers 63](#_Toc201972869)

[Table 9: Final dataset selection scoring metrics 67](#_Toc201972870)

[Table 10: Metrics for best model 70](#_Toc201972871)

[Table 11: Comparison with Rodriguez et al. [42] study 71](#_Toc201972872)

[Table 12: Training loss evolution for PH3-based models used for live-inference 77](#_Toc201972873)

LIST OF ACRONYMS AND ABBREVIATIONS

|  |  |  |
| --- | --- | --- |
| ASL |  | American Sign Language |
| BERT |  | Bidirectional Encoder Representations from Transformers |
| CNN |  | Convolutional Neural Network |
| DM |  | Direction of Motion |
| GRU |  | Gated Recurrent Unit |
| HC |  | Hand Configuration |
| HM |  | Hand Motion |
| HO |  | Hand Orientation |
| KNN |  | K-Nearest Neighbors |
| LSTM |  | Long Short-Term Memory |
| MSL |  | Mexican Sign Language |
| NMC |  | Non-Manual Cues |
| PA |  | Place of Articulation |
| PCA |  | Principal Component Analysis |
| PH1 |  | Phase 1 (Landmark extraction) |
| PH2 |  | Phase 2 (Geometric Transformations) |
| PH3 |  | Phase 3 (Dimensionality Reduction) |
| RBF |  | Radial Basis Function |
| RGB |  | Red, Green, Blue (color channels in images) |
| RNN |  | Recurrent Neural Network |
| SVM |  | Support Vector Machine |
| UMAP |  | Uniform Manifold Approximation and Projection |

# 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]. Mexican Sign Language (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, a reflection of the language’s reach relative to both the hard-of-hearing population (13 %) and the general population (0.2 %) [3], [4], and of the isolation that Deaf people sometimes live in. Enabling communication between Deaf and hearing people is key to fostering inclusion in education, the workplace, 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 routine 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 author of this work is 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 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 [11], or jointly [7], [12].

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 in [10], our approach treats each sign at the lexical level as a composition of six minor units:

* Hand Configuration (HC)
* Place of Articulation (PA)
* Hand Motion (HM)
* Direction of Motion (DM)
* Hand Orientation (HO)
* Non-Manual Cues (NMC)

We apply a complex data preparation methodology to extract the features units that concern our videos, and use it to develop what is, to our knowledge, the first highly efficient, real-time recognition unified system for both static and dynamic alphanumeric MSL signs that runs on consumer-grade hardware. Throughout this process, we also conduct experiments to evaluate the hypothesis that preprocessing focused on distilling the six minor units of MSL (HC, PA, HM, DM, HO, NMC) 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 Mexican Sign Language. 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

Mexican Sign Language (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, syntax, and lexicon, distinct from spoken Spanish [10]. Mexican Sign Language was officially recognized as a national language by the mexican government 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 Mexican Sign Language 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 as 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. propose comprehensive guidelines to describe signs as combination of the following features in [10]:

* Hand Configuration (HC)
* Place of Articulation (PA)
* Hand Motion (HM)
* Direction of Motion (DM)
* Hand Orientation (HO)
* Non-Manual Cues (NMC)

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 are limited to single-hand sign recognition. Although a path extending the system to bimanual sign recognition is easy to envision using the same tools we already use, it would require a complete reconfiguration of our pipeline. This is the main limitation in terms of potential for rapidly extending the use 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 is a complete listing of the options in use for Mexican Sign Language. 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 1. 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 features is unrealistic given the current publicly available MSL datasets. This research focuses on hand configuration for four 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 must have a specific hand shape. 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 Machine Learning 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 in [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, V. 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 1).

|  |
| --- |
| Figure 3: Hand orientations as shown in [10] |



For hand placement, there was minimal coverage, as 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, 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 in the depth axis.

We observe somewhat greater variation in 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 machine learning.

Early research in the field often approached the problem using statistical models. For example, in 1995, [17] achieved real-time recognition of American Sign Language word signs using Hidden Markov Models. 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, Hidden Markov Models 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 machine learning models such as Support Vector Machines (SVMs) and K-Nearest Neighbors (KNN) [6]. These models offered lower computational cost and a higher potential for scalability as compared to Hidden Markov Models but lost the intrinsic sequential component that helped HMMs to deal 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 machine learning 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 [20]. Unlike traditional machine learning 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, 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) [21], [22] became especially prominent in Sign Language Recognition in the early 2010s, thanks to their ability to exploit spatial structures in images [23]. 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 [20]. 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 meaningful. 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 [20].

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

Sign Language Recognition projects using Convolutional Neural Networks directly to predict classes have obtained excellent results and have repeatedly advanced accuracy standards in the field [5] [24]. 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 Machine Learning [20]. 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 is [23], where first a CNN is 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 in [16], the authors use the OpenPose library [25] to extract features and then feeds them into a LSTM model. Other cases [11] like this make use of the Google MediaPipe framework which provides pre-trained CNN models for problems such as hand landmarking, pose detection, and face landmarking [26] [27]. The hand landmarking model extracts three dimensional coordinates for the 21 points which can be seen in Figure 2 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 (RNNs) [28] complement Convolutional Neural Networks 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 Long Short-Term Memory (LSTM) [29] and Gated Recurrent Units (GRU) , 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, inference can be slower, leading researchers to explore Transformer architectures.

Transformers, introduced in 2017 by Vaswani et al. [30], 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 or frame) 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, even if they are far apart, is what gives Transformers a significant advantage over RNNs, which process the sequence step-by-step and struggle to maintain long-range dependencies. In Sign Language Recognition (SLR), this is especially valuable for tasks like continuous sign language recognition, where signs depend on the overall context of the sequence. For example, Mendoza López et al. [31] applied a multitask transformer model using optical flow to improve accuracy in recognizing complex sign sequences. 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 such as compact BERT [32], [33] models DistilBERT [34] and have been developed to reduce size and inference time while retaining much of the original model's performance.

## Mexican Sign Language Recognition

Research on Mexican Sign Language (LSM) 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) [35] 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 [36] 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 Dynamic Time Warping [37] to classify dynamic signs, representing early structured MSL recognition, and achieving an accuracy of 98.6%. Mejía-Pérez et al. [12] later applied RNNs [28] 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 [26] and trained multiple deep learning models, including Transformers [30], 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. [38] achieved between 97.3% and 99.4% accuracy on alphabet recognition using MediaPipe hand landmarks combined with SVM [39] and Random Forest [40] 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. developed a dataset covering 100 of the most frequently used MSL signs, totaling 5,000 videos. Their best model, a CNN [21] combined with a BiLSTM [41], 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], [42], who created and published a dataset of MSL alphabet and number signs [43], [44] 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 [45], SVM [39], LSTM [29], and GRU [46], 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 [30] architectures, and evaluate alternative data-preprocessing and augmentation strategies.

Table : State of the Art MSL systems comparison

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Study** | **Lexicon** | **Input capture** | **Method** | **Best accuracy** |
| García-Bautista et al. (2017) [13] | 20 words | Kinect with depth | DTW | 98.6% |
| Mejía-Pérez et al. (2022) [12] | Digits (0-9) | Depth and RGB | RNN | 97.0% clean, 90.3% noisy |
| González-Rodríguez et al. (2024) [9] | 10 multi-word phrases | Kinect and RGB | MediaPipe + RNN, LSTM, Transformer | 98.8% |
| Marquez et al. (2025) [35] | Alphabet | RGB | CNN | 92% |
| Cataño et al. (2024) [38] | Alphabet | RGB | MediaPipe + SVM, Random Forest | 99.4% |
| Martínez-Sánchez et al. (2023) [7] | 100 words | RGB | CNN + BiLSTM | 99% |
| Rodríguez et al. (2025) [11] | Alphanumerc | RGB | MediaPipe +, SVM, GRU | 94.1% static, 84.7% dynamic |
| Present work | Alphanumeric | RGB | See Methodology section | See Results section |

# 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 lexical structure of signs in Mexican Sign Language, feature extraction and representation, feature reduction, neural network architectures used in Sign Language Recognition, and the evaluation metrics used to compare models.

## Lexical 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 Mexican Sign Language (MSL), Cruz Aldrete et al. [10] propose that each sign can be described using six components:

* Hand Configuration (HC)
* Place of Articulation (PA)
* Hand Motion (HM)
* Direction of Motion (DM)
* Hand Orientation (HO)
* Non-Manual Cues (NMC)

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 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 [22] and [47].

## Landmark Extraction with MediaPipe

Instead of working directly with image pixels, this project uses landmark coordinates as input. MediaPipe is a tool 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 small set of 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 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). 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 details on the theoretical foundations, see [48].

For more complex data structures where linear assumptions fall short, Uniform Manifold Approximation and Projection (UMAP) [49] 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

Recurrent Neural Networks (RNNs) are designed to handle sequential data, where the order of inputs matters [28]. 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 (SLR). In the context of SLR, 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 [29]. 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 [29].

Long Short-Term Memory (LSTM) [29] and Gated Recurrent Units (GRU) [50] 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 because they can capture temporal dependencies in dynamic signs, which is crucial for accurate recognition [42], [9].

Transformers [30] 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 into the input. This design allows Transformers to model long-range dependencies more effectively than RNNs while also enabling faster training through parallelism.

BERT (Bidirectional Encoder Representations from Transformers) 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 [32].

## 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 [51].

* **Accuracy**:
  + The proportion of correct predictions over the total number of predictions:
* **Precision (per class)**:
  + For a given class :
    - 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 :
    - where is the number of times class was the correct label but not predicted
* **Macro-Averaged Precision and Recall**:
  + where is the number of classes.
* **F1 Score (per class)**:
  + The harmonic mean of precision and recall for class :
* **Macro F1 Score**:
  + The average F1 score across all classes:
* **Top-2 Accuracy**:
  + The proportion of times the true class is among the top 2 predicted classes:
* **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:
    - Variance of accuracy:
    - Z-score:

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 in [10], we base our methodology on the idea that, at the word level, MSL signs can be divided into 6 minor units:

* Hand Configuration (HC)
* Place of Articulation (PA)
* Hand Motion (HM)
* Direction of the Motion (DM)
* Hand Orientation (HO)
* Non-Manual Cues (NMC)

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 1.

|  |
| --- |
| Figure : Methodology overview |

## Raw dataset

Table : Samples from the alphanumeric MSL dataset

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

We obtained a publicly shared MSL dataset of 11 signers, each performing the 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], [43], [44]. The recordings isolate single static and dynamic signs under controlled conditions, providing balanced coverage across signers, classes, and handedness. Table 2 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 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 |

We leverage 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 dialect, as can be seen in Table 1. With context, these two signs are generally immediately distinguishable. However, in our isolated sign language context, 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.

|  |
| --- |
| Figure : Data Preparation |

## (PH1) Video to landmarks

## Purpose of using landmarks instead of images

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

* hand configuration
* hand orientation
* hand movement
* 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 [27] [26]. MediaPipe Hands returns 21 three-dimensional hand landmarks, which can be seen in Figure 6, as well as handedness labels, and confidence scores; MediaPipe Pose returns a well-dispersed set of pose landmarks, which can be seen in Figure 7.

|  |  |
| --- | --- |
| Figure : MediaPipe Hand Landmarks | Figure : 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. Additionally, the model was partially trained on computer generated 3D images of hands, 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 in under 10 ms per frame on CPU—critical for live inference.

## Use of MediaPipe for this project



|  |
| --- |
| Figure : Phase 1: Landmark Extraction |

A diagram summarizing this section can be seen in Figure 8.

We read each video using Python's OpenCV [52] 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 performed. 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 [42] used for this project, M. E. Rodriguez, used a strategy in a similar project 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 second 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 had 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 offer a proxy to “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 7). 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 works with only 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 is not necessarily related 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 it’s 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 2, 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 was 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) Landmark Transformations

|  |
| --- |
| Figure : Phase 2: Landmark Transformations |

###### As mentioned previously, as far as the hand goes, MSL signs consist of four core components—hand configuration, hand orientation, hand placement, and hand movement [1]—all of which we already encode via raw landmark coordinates. However, these components intermix in the original camera frame. In PH2, we isolate seek to isolate each component. The diagram in Figure 9 summarizes the following descriptions.

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

###### To obtain 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-centered frame of reference.

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

A step-by-step pseudocode version of this transformation process is provided in Appendix A for reference and clarity.

## 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 K Nearest Neighbors 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.

##### 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 first 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, 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 : 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 n×n kernel matrix, a randomly sampled subset of m≪n “landmark” points is selected, with which only the n×m and m×m 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 O(n2) to O(nm). 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, n) combination yields a new dataset fork of the same 69572 rows and 8+n columns, where 8 corresponds to the same video tagging, labeling and frame indexing columns as before, and n 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 the per-frame data that only includes rows corresponding to active hands, and we colored by the sign class**
* **the third corresponds to the 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 here some of the more insightful ones.**

**The visualizations colored by sign are much harder to interpret simply because of 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 11 **and** Figure 12 **show the results for the per-frame data with or without the PH2 transformations, respectively.**

|  |  |
| --- | --- |
| Figure : 3-component reduction of the PH2-processed per-frame data using PCA with a cosine kernel and coloring by sign | Figure : 3-component reduction of the 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 could only take two values, were much easier to interpret.**

**In** Table 3**, 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 Radial-Based Function (RBF) kernels stand out as the best.**

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

|  |  |
| --- | --- |
| Figure : Using a cosine kernel | Figure : Using a polynomial kernel |
| Figure : Using a radial basis kernel | Figure : Using a sigmoid kernel |

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

|  |  |
| --- | --- |
| Figure : Using a cosine kernel | Figure : Using a polynomial kernel |
| Figure : Using a radial basis kernel | Figure : 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 some 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 21 **and** Figure 22**,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 23 **and** Figure 24**, we can see that the axial nature is “lost”. Intuitively, time being the root of these continuous progressions along lines is also an appealing interpretation.**

|  |  |
| --- | --- |
| Figure : 3-component reduction of the PH2-processed per-frame data using PCA with a radial-based function kernel and coloring by sign | Figure : 3-component reduction of the non-PH2-processed per-frame data using PCA with a radial-based function 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 : 3-component reduction of the PH2-processed per-video data using PCA with a radial-based function kernel and coloring by sign | Figure : 3-component reduction of the non-PH2-processed per-video data using PCA with a radial-based function kernel and coloring by sign |

##### **On a separate point, 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 slight difference in the general shape that the reduced data formed. The most noticeable is shown in** Figure 25 **and** Figure 26**, 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 : 3-component reduction of the PH2-processed per-frame data using UMAP and coloring by sign | Figure : 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, 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 5. 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” [33], and in [34]. 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** [53] | 2 | 128 | 4.43 million |
| **bert-mini** [54] | 4 | 256 | 11.3 million |
| **bert-small** [55] | 4 | 512 | 29.1 million |
| **bert-medium** [56] | 8 | 512 | 41.7 million |
| **DistilBERT** [57] | 6 | 768 | 66 million |

Table 6: Size comparison of different BERT inspired architectures as seen in [53]

###### 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 K-Nearest Neighbors models—to identify the most promising preprocessing and reduction settings 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 phase. With this two-staged approach we are able to focus our resources on the dataset-model configurations that appear to have the most potential. Figure 27 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 n components, with n going from 1 to 15, making for 15 options
  + PCA and one of four kernels (Polynomial, Cosine, Sigmoid or RBF) for n components, with n going from 1 to 15, making for 4\*15 = 60 options



* + UMAP for n components, with n 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 28 shows the strategy used to do so.

|  |
| --- |
| Figure : 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 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 for number of classes: 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.

## Fit 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 K-Nearest Neighbors models to the data. However, once again to be computationally resource-conscious, 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 K-Nearest Neighbors 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 n-class problem with N total examples, let be the number of examples whose true label is i and whose predicted label is j. Then



The accuracy (Equation 2) is the proportion of correct predictions:



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 i is equally likely to be the true label:



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



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



Now, assuming uniform random guessing among n classes:

1. Each ordered pair (true=i, guess=j) has probability



1. Correct predictions occur when i=j. There are n such diagonal events, each of probability 1/n², so  
   .



Hence the expected (mean) accuracy under random guessing is



Under random guessing, each of the N examples has probability 1∕n of being correctly classified. We model the total number of correct predictions, C, as a Binomial(N, 1∕n) random variable.

1. Expected value of C



1. Variance of C



1. And we have that accuracy is C / N, therefore the variance of accuracy is:



1. And



1. For an observed accuracy Â, the Z-score is



## 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 : 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 29. 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 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 . This however, is somewhat contradictory to the linear growth we are extrapolating from these 18 datapoints. We believe the next point could explain this incongruity
* Models generally become better the more data we have to train them on, and it seems our models still have room to grow in this area. Therefore, if in the future larger datasets were made available for MSL, it seems the methodology explored here should scale well.

|  |
| --- |
| Figure : 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 30, 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 samples of the per-frame data. This fact combined with the general general trend we saw for number of classes suggests to us that once again there 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. This 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

|  |
| --- |
| Figure : 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 on our data. In Figure 31, we compare data that went through geometric transformations which 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 Mexican Sign Language may be being lost in our transformations.

Abstractly, our transformations try to isolate hand configuration, hand placement and hand orientation from each other. It is possible that doing so is a mistake. This result does not necessarily contradict the description that Mexican Sign Language can be decomposed into 6 minor units given by [10], as the description does not expand on how these 6 components relate to each other. Nevertheless, this result is the first of several we observe in this research that does suggest that the 6 components are not independent.

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 will make decisions based on the results we obtain in this phase of analysis.

The first result we look into is seen in Figure 32, 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 : Evolution over the number of classes of the significance scores obtained for data passed through PH3 (orange) vs data that did not (blue) |

|  |
| --- |
| Figure : 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 33, using no reducer, PCA, PCA combined with a kernel, or UMAP, do not have 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 decided that moving forward we would no longer use it.

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

After that, we studied the different kernels we could use to combine with PCA. Figure 34 confirms what we observed in our 3-dimensional visualizations of data: using a cosine kernel is the best option. However, it performs very similarly to using no kernel at all. Because of this, we decided to move forward with only one PH3 reduction technique: PCA with no kernel.

Interestingly, the Radial-Based Function kernel performed far worse than all others. This indicates that the axial structure the data showed in our 3-dimensional visualizations earlier had little relation to the intrinsic structure of MSL which KNN models extract. This supports our theory that it could be a time-dependent feature.

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 35 shows that PCA generally performed better as the number of components went up.

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

## 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 the same results. Table 6 shows the best results for a selection of choices:

Table : 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.

## Best KNN Sign Recognition

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 8 the best results we obtained when claissifying the 37 alphanumeric classes using KNN, with the different data preparation combinations we explore.

Table : 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 8 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 5.

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

* Optimizer: AdamW [58]
* Learning rate:
* Loss function: Cross Entropy Loss [59]

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.

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.

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

As we can see in Figure 36, the bert-mini architecture [54] previewed the best results in this exploration, and moving forward we discard the study of all other architectures.

|  |
| --- |
| Figure : 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 37, 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

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

* 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 Recurrent Neural Networks because they performed better at long-range dependency tracking [30]. 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 38 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.

## 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 7 shows the results we obtained.

Table : 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.

## Best model finetuning

The chosen dataset–BERT-mini pair undergoes extended training for 2.5× the number epochs used Section 4.4.4. After some exploration, the final configurations used are:

* Dataset
  + Data unit: frame
  + PH2 geometric transformations: not applied
  + PH3 dimensionality reduction: not applied
* Model
  + Pre-trained model: bert-mini-uncased [54]
  + Optimizer: AdamW [58]
  + Learning rate:
  + Loss function: Cross Entropy Loss [59]
  + 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 8.

Table : Metrics for best model

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 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 39.

|  |
| --- |
| Figure : Training loss evolution for best model |

After testing, the confusion matrix was computed and can be seeing in Figure 40.

|  |
| --- |
| Figure : Confusion matrix for best model |

## Discussion

## Comparing to similar studies

This study closely resembles that of Rodriguez et al. [42], 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 Mario 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 9 our testing metrics to those of Rodriguez et al.

Table : Comparison with Rodriguez et al. [42] study

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Macro F1 score | Macro Precision | Macro Recall |
| (Static signs) GBL [42] | 77.04% | 76.85% | 76.82% | 76.19% |
| (Static signs) SVM [42] | 94.07% | 93.73% | 94.25% | 93.56% |
| (Dynamic signs) LSTM [42] | 79.59% | 80.31% | 80.93% | 80.28% |
| (Dynamic signs) GRU [42] | 84.69% | 86.01% | 86.00% | 85.57% |
| (Present work) (37 classes) BERT-mini, per frame,  no PH2 | 97.40% | 97.40% | 97.42% | 97.40% |

As we can see, our best 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 best 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 machine learning 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 best 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 best model works on a per-frame basis, it has not 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 40 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 43 and Figure 44 show the alphanumeric signs once again, thanks to the work of IncluSor [15].

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

The most noticeable drops in diagonal density occur for the “LL” and “RR” signs. 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 1); {6, 8} have the back of the hand facing the camera but the fingers pointing inwards to the side (number 9 in Figure 1); 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 1). This is visible in Figure 44.
  + 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 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 40 relating to: “A” vs “B”; “LL” vs “RR”; “C” vs “O”; “6” vs “8”; and many others.

Additionally, it seems that some of the confusions can be explained by similarities in hand configuration. See for example in Figure 40 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 in [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.

## Live-inference system

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 9. However, this model performs quite poorly as the backbone of our live-inference system where it faces data in a setting that is different, though not obviously so, from the dataset on which it was trained and tested. We believe that the model is overfit on some characteristics that appear in both the training and testing data, but is not generally present in other settings.

In our exploratory use of the live-inference system based on this “best model”, we notice that the prediction sticks to the same 2 or 3 signs independently of the sign being performed, barely seeing any change even between transitions. This indicates to us that it is too sensible to some parameters in our setting which it was not exposed to in the training or testing data, and which it is severely negatively affected by.

This is an especially disapointing outcome given that we expected our system to be quite robust due to it’s use of MediaPipe as it’s foundation, a framework trained on an incredibly varied set of data [26], and which itself does perform well in its role of landmark detection. However the poor performance of our system using this model makes us believe that this inherited robustness blinded us to the dangers of overfitting in other phases of our process. We believe that the low-variability is coming not from the “visual” settings, but instead perhaps from the “anatomical” ones. Perhaps the model is adapting itself to the signers bodies—for which we extract representations using MediaPipe—too much. Therefore, because these same people appear in both the training and testing data, we obtain deceptively good results, which do not accurately reflect how the model will perform with other signers. To test this theory, we could try training models on only a certain subset of the signers, and then testing on the others, or make use of other MSL datasets including alphanumeric signs and testing with that.

After noticing the poor performance of our model for live inference, we decided to try the models trained using data that went through different processing paths. Of these, the models based on data which used either PH2 geometric transformations or PH3 dimensionality reduction techniques were the best in the sense that they were more responsive to changes between signs. Contrary to the raw-landmark models (both per-frame or per video), the models which were trained on data which was somehow preprocessed were more sensitive to the particularities of each sign. Of these the best were the dimensionality-reduced variants (both per-frame and per-video), but even these were not highly accurate. Nevertheless, the predictions made often had some relation to the true class of the sign being articulated, either through a similar hand configuration, hand orientation, or by being contiguous in the alphabet or numbering order (an indication of signs leaking into each other’s videos in the training dataset). Thus, we believe the scores obtained for those models (with PH3, no PH2, either through per-frame or per-video data), are more representative of the system obtained in this project. The evolution of the training loss for each of these PH3-based models can be seen in Table 10, and the confusion matrices obtained with the testing data can be seen in Figure 43 and Figure 44. Evidently, neither model performs as highly when used for live-inference as it did for the testing data, but the traces of the powerful model behind the system is clearer.

Table : Training loss evolution for PH3-based models used for live-inference

|  |  |
| --- | --- |
| Figure : Per frame data | Figure : Per video data |

|  |
| --- |
| Figure : Confusion matrix for the per-frame, PH3-based model used for live-inference |

|  |
| --- |
| Figure : Confusion matrix for the per-video, PH3-based model used for live-inference |

Our hypothesis is that possibly some of the features the raw-landmark models were overfitting onto were lost in the dimensionality reductions, which inadvertently made our model more robust. However, a methodology to test this hypothesis is not clear for the authors of this work.

It is also interesting to note that the per-video models, though somewhat worse overall for live-inference, did noticeably respond to movement patterns more effectively than the per-frame models, showing the latent potential of this approach.

In conclusion, finding a combination of the data preparation processes proposed in this work, lightweight and pre-trained transformer architectures and training hyperparameters which allow us to use this per-video approach while obtaining similarly highly performant models as our “best model” seems within reach for a more experienced machine learning tuner.

## Comparison between KNN and BERT-mini

A useful comparison can be made between the best-performing KNN models (Table 8) and the results obtained for BERT-Mini (Table 9and Table 10). 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 Mexican Sign Language 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 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 conclusion: 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 Mexican Sign Language 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, making it a practical foundation for future assistive 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 made 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 Mexican Sign Language (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.

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.

Finally, 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 [60] 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.

# 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] | F. Chollet, Deep Learning with Python, Second Edition, Manning Publications, 2021. |
| [21] | 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. |
| [22] | 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]. |
| [23] | O. Koller, S. Zargaran, H. Ney and R. Bowden, "Deep Sign: Enabling Robust Statistical Continuous Sign Language Recognition via Hybrid CNN-HMMs". |
| [24] | 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. |
| [25] | 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]. |
| [26] | 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. |
| [27] | Google, "Hand landmarks detection guide," Google, 21 May 2024. [Online]. Available: https://ai.google.dev/edge/mediapipe/solutions/guide. [Accessed 28 Nov 2024]. |
| [28] | J. L. Elman, "Finding Structure in time," *Cognitive Science,* vol. 14, no. 2, pp. 179-211, 1990. |
| [29] | S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation,* vol. 9, no. 8, pp. 1735-1780, 1997. |
| [30] | A. Vaswani et al., "Attention is all you need," in *Advances in Neiral Information Processing Systems (NeurIPS)*, 2017. |
| [31] | 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]. |
| [32] | Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *NAACL,* 2019. |
| [33] | 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]. |
| [34] | 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]. |
| [35] | 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. |
| [36] | 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]. |
| [37] | 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. |
| [38] | 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. |
| [39] | C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning,* vol. 20, no. 3, pp. 273-297, 1995. |
| [40] | L. Breiman, "Random forests," *Machine Learning,* vol. 45, no. 1, pp. 5-32, 2001. |
| [41] | M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.,* vol. 45, no. 11, pp. 2673-2681, 1997. |
| [42] | 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. |
| [43] | 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]. |
| [44] | 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]. |
| [45] | K. Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, 2012. |
| [46] | K. Cho et al., "Learning phrase representations using RNN encoder-decoder fir statistical machine translation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014. |
| [47] | I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, Cambridge, MA, USA: MIT Press, 2016. |
| [48] | I. T. Jolliffe, Principal Component Analysis, New York, NY, USA: Springer, 2002. |
| [49] | McInnes et al., "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction," *Journal of Open Source Software,* 2018. |
| [50] | 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]. |
| [51] | C. D. Manning, P. Raghavan and H. Schütze, Introduction to Information Retrieval, Cambridge University Press, 2008. |
| [52] | OpenCV, "OpenCV - Open Computer Vision Library," [Online]. Available: https://opencv.org. [Accessed 28 Nov 2024]. |
| [53] | T. Tran, "gaunerst/bert-tiny-uncased," 2024 October 2024. [Online]. Available: https://huggingface.co/gaunernst/bert-tiny-uncased. [Accessed 22 June 2025]. |
| [54] | T. Tran, "gaunernst/bert-mini-uncased," 25 October 2024. [Online]. Available: https://huggingface.co/gaunernst/bert-mini-uncased. [Accessed 22 June 2025]. |
| [55] | T. Tran, "gaunernst/bert-small-uncased," 4 January 2025. [Online]. Available: https://huggingface.co/gaunernst/bert-small-uncased. [Accessed 22 June 2025]. |
| [56] | T. Tran, "gaunernst/bert-medium-uncased," 04 January 2025. [Online]. Available: https://huggingface.co/gaunernst/bert-medium-uncased. [Accessed 22 June 2025]. |
| [57] | H. F. t. team, "distilbert/distilbert-base-uncased," [Online]. Available: https://huggingface.co/distilbert/distilbert-base-uncased. [Accessed 22 June 2025]. |
| [58] | PyTorch, "AdamW," [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.optim.AdamW.html. [Accessed 23 June 2025]. |
| [59] | PyTorch, "CrossEntropyLoss," [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html. [Accessed 23 June 2025]. |
| [60] | V. Shiv and C. Quirk, "Novel Positional Encodings to Enable Tree-Based Transformers," *Advances in Neural Information Processing Systems,* 2019. |
| [61] | 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. |
| [62] | 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. |
| [63] | 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. |
| [64] | Scikit-Learn, "PCA," [Online]. Available: https://scikit-learn.org/dev/modules/generated/sklearn.decomposition.PCA.html. [Accessed 28 Nov 2024]. |
| [65] | Scikit-Learn, "KNeighborsClassifier," [Online]. Available: https://scikit-learn.org/dev/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. [Accessed 28 Nov 2024]. |
| [66] | T. Tran, "google-bert/bert-base-uncased," Google, [Online]. Available: https://huggingface.co/google-bert/bert-base-uncased. [Accessed 22 June 2025]. |
| [67] | OpenAI, "ChatGPT," OpenAI, [Online]. Available: https://chat.openai.com/. [Accessed 24 June 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 : No PH2, active hand detection, PCA, no kernel | Figure : No PH2, active hand detection, PCA, cosine kernel |
| Figure : No PH2, active hand detection, PCA, polynomial kernel | Figure : No PH2, active hand detection, PCA, RBF kernel |
| Figure : No PH2, active hand detection, PCA, sigmoid kernel | Figure : No PH2, active hand detection, UMAP |
| Figure : No PH2, per-frame sign recognition, PCA, no kernel | Figure : No PH2, per-frame sign recognition, PCA, cosine kernel |
| Figure : No PH2, per-frame sign recognition, PCA, polynomial kernel | Figure : No PH2, per-frame sign recognition, PCA, RBF kernel |
| Figure : No PH2, per-frame sign recognition, PCA, sigmoid kernel | Figure : No PH2, per-frame sign recognition, UMAP |
| Figure : No PH2, per-video sign recognition, PCA, no kernel | Figure : No PH2, per-video sign recognition, PCA, cosine kernel |
| Figure : No PH2, per-video sign recognition, PCA, polynomial kernel | Figure : No PH2, per-video sign recognition, PCA, RBF kernel |
| Figure : No PH2, per-video sign recognition, PCA, sigmoid kernel | Figure : No PH2, per-video sign recognition, UMAP |
| Figure : PH2 transformed, active hand detection, PCA, no kernel | Figure : PH2 transformed, active hand detection, PCA, cosine kernel |
| Figure : PH2 transformed, active hand detection, PCA, polynomial kernel | Figure : PH2 transformed, active hand detection, PCA, RBF kernel |
| Figure : PH2 transformed, active hand detection, PCA, sigmoid kernel | Figure : PH2 transformed, active hand detection, UMAP |
| Figure : PH2 transformed, per-frame sign recognition, PCA, no kernel | Figure : PH2 transformed, per-frame sign recognition, PCA, cosine kernel |
| Figure : PH2 transformed, per-frame sign recognition, PCA, polynomial kernel | Figure : PH2 transformed, per-frame sign recognition, PCA, RBF kernel |
| Figure : PH2 transformed, per-frame sign recognition, PCA, sigmoid kernel | Figure : PH2 transformed, per-frame sign recognition, UMAP |
| Figure : PH2 transformed, per-video sign recognition, PCA, no kernel | Figure : PH2 transformed, per-video sign recognition, PCA, cosine kernel |
| Figure : PH2 transformed, per-video sign recognition, PCA, polynomial kernel | Figure : PH2 transformed, per-video sign recognition, PCA, RBF kernel |
| Figure : PH2 transformed, per-video sign recognition, PCA, sigmoid kernel | Figure : PH2 transformed, per-video sign recognition, UMAP |

**APPENDIX C. Confusion matrices and training loss evolution for the final candidate BERT-mini models**

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

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

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

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

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

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

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

**APPENDIX D. Confusion matrices for the best KNN sign classifier**

|  |
| --- |
| Figure : 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)