



EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE CULIACÁN



**Gestura: software interprete del lenguaje de señas para
personas con discapacidad auditiva y del habla en Culiacán.**

TESIS

OBTENER EL GRADO DE
INGENIERO EN SISTEMAS COMPUTACIONALES

POR:

**Rosas León Ramón Alberto
Verdugo Reyes Jesús Javier**

DIRECTOR DE TESIS:
Bustillos Martínez Arcelia Judith

CULIACÁN, SINALOA

20 DE JUNIO DE 2025



Instituto Tecnológico de Culiacán

DIVISION DE ESTUDIOS PROFESIONALES
ASUNTO: **Autorización de Impresión**

Culiacán, Sinaloa, 23/09/2025 19:22

C. RAMÓN ALBERTO ROSAS LEÓN Y JESUS JAVIER VERDUGO REYES
PASANTE(S) DE LA CARRERA DE INGENIERÍA EN SISTEMAS COMPUTACIONALES
P R E S E N T E.

De acuerdo con fallo emitido por la Comisión Revisora Integrada por:

Cédula Profesional

Presidente:	Maestría en Ciencias en Ciencias Computacionales Arcelia Judith Bustillos Martínez	13118410
Secretario:	Doctorado en Ciencias en Ingeniería Eléctrica Opción en Sistemas Computacionales Héctor Rodríguez Rangel	9972835
Vocal:	Maestría en Informática Aplicada Rogelio Prieto Alvarado	13976653

Y considerando que reúne los requisitos establecidos en la Opción: **TITULACIÓN INTEGRAL: TESIS PROFESIONAL "GESTURA: SOFTWARE INTERPRETE DEL LENGUAJE DE SEÑAS PARA PERSONAS CON DISCAPACIDAD AUDITIVA Y DEL HABLA EN CULIACÁN"**, damos a usted nuestra autorización para que proceda a imprimir su trabajo profesional.

Sin otro particular, quedo de usted.

ATENTAMENTE
Excelencia en Educación Tecnológica®



Educación
Secretaría de Educación Pública



TECNOLOGICO
NACIONAL DE MEXICO

INSTITUTO TECNOLÓGICO DE CULIACÁN

LIC. CONCEPCION MENDOZA ROSALES

JEFA DE LA DIVISIÓN DE ESTUDIOS PROFESIONALES

DEPARTAMENTO DE
DIVISIÓN DE ESTUDIOS PROFESIONALES



2025
Año de
La Mujer
Indígena

Juan de Dios Bátis 310 Pte. Col. Guadalupe C.P. 80220
Culiacán, Sinaloa. Tel. 667-454-0100
tecnm.mx www.culiacan.tecnm.mx



Hoja de firmas

Dedicatoria

Ramón Alberto Rosas León

A mis padres, que siempre creyeron que podía lograr esto aunque yo no estuviera tan seguro. Por aguantar mis frustraciones cuando el código no funcionaba y por preguntarme cómo iba el proyecto aunque no entendieran ni la mitad de lo que les explicaba.

A mis amigos, por seguir invitándome a salir sabiendo que probablemente diría "no puedo, tengo que trabajar en la tesis". Y por celebrar conmigo cuando finalmente las cosas salían bien.

Jesús Javier Verdugo Reyes

A mi familia, que me enseñó que los proyectos importantes requieren tiempo y dedicación. A mis padres, por darme las herramientas para enfrentar retos como este y por respetar mis horarios de trabajo, incluso cuando significaba cenas familiares sin mí.

A mis amigos, que entendieron que este semestre iba a ser diferente y que me esperaron pacientemente hasta que tuviera tiempo libre otra vez.

A Ramón, porque resultó que sí se puede hacer buen trabajo en equipo cuando ambos se comprometen en serio. Gracias por mantener la calma cuando yo me estresaba.

Agradecimientos

Ramón Alberto Rosas León

Quiero agradecer al Dr. Héctor Rodríguez Rangel y a la Mtra. Arcelia Judith Bustillos Martínez, cuya guía y apoyo fueron clave en este proyecto. Sus consejos enriquecieron mi trabajo y me motivaron a superar cada desafío.

También al Instituto Tecnológico de Culiacán, por proporcionar el espacio y los recursos necesarios para mi formación.

A mi familia, por su paciencia y confianza inquebrantable, y a mis compañeros y amigos, por su apoyo y colaboración en cada etapa.

Un agradecimiento especial a mi compañero de tesis, Jesús Javier Verdugo Reyes, por su compromiso, esfuerzo y colaboración en este camino.

A todos, gracias por ser parte de este logro.

Jesús Javier Verdugo Reyes

Expreso mi más sincero agradecimiento al Instituto Tecnológico de Culiacán, institución que me brindó la oportunidad de formarme profesionalmente y desarrollar mis habilidades.

A mi asesora interna de tesis, Mtra. Arcelia Judith Bustillos Martínez, por su invaluable guía en la estructura y documentación del trabajo, y al Dr. Héctor Rodríguez Rangel, mi asesor externo, por su conocimiento y experiencia en el desarrollo del proyecto.

A mi padre, Arturo Verdugo Ezpinoza, y a mi madre, Antonia María Reyes Silva, por ser mi mayor inspiración y por su amor incondicional. A mis hermanos, Arturo Verdugo Reyes y Sergio Antonio Verdugo Reyes, por su compañía y aliento constante.

Un agradecimiento muy especial a mi tía, Sara Isabel Reyes Silva, quien me abrió las puertas de su hogar durante mi etapa estudiantil, y a mi cuñada, Ana Adelina Carrillo, por su comprensión y motivación.

A mi compañero de tesis, Ramón Alberto Rosas León, por su compromiso, dedicación y excelente trabajo en equipo durante todo este proceso.

Declaración de autenticidad

Por la presente declaro que, salvo cuando se haga referencia específica al trabajo de otras personas, el contenido de esta tesis es original y no se ha presentado total o parcialmente para su consideración para cualquier otro título o grado en esta o cualquier otra Universidad. Esta tesis es resultado de mi propio trabajo de investigación y desarrollo del sistema "Gestura" para interpretación de Lenguaje de Señas Mexicano en tiempo real, y no incluye nada que sea resultado de algún trabajo realizado en colaboración, salvo que se indique específicamente en el texto. El desarrollo del software, la recolección del dataset, el entrenamiento del modelo de inteligencia artificial, las pruebas con usuarios y la documentación de resultados fueron realizados íntegramente por el suscrito durante el período de residencias profesionales.

Ramon Rosas,

Javier Verdugo

Culiacán, Sinaloa, México, 2024

Resumen

Este trabajo trata sobre desarrollar e implementar un sistema basado en inteligencia artificial para interpretar el Lenguaje de Señas Mexicano (LSM) en tiempo real. La idea principal era facilitar la comunicación entre personas sordas y oyentes cuando van a hacer trámites de ventanilla en instituciones públicas y privadas de Culiacán, Sinaloa. Todo esto surgió porque hay barreras comunicativas que enfrentan aproximadamente 6,771 personas con discapacidad auditiva en la ciudad - gente que encuentra dificultades para acceder a servicios esenciales simplemente porque no existen herramientas tecnológicas inclusivas.

Desarrollamos el sistema "Gestura" usando una metodología de desarrollo tecnológico experimental que duró 16 semanas. Para extraer keypoints corporales usamos MediaPipe, y para clasificar los gestos empleamos redes neuronales bidireccionales de memoria a largo y corto plazo (BiLSTM). La arquitectura incluye un algoritmo de segmentación automática que detecta cuándo empiezan y terminan las señas sin que alguien tenga que intervenir manualmente. También procesamiento multimodal de audio que convierte voz a texto y texto a voz, más una interfaz de usuario nativa que desarrollamos en PyQt6.

Para el dataset personalizado recolectamos 1,400 muestras correspondientes a 28 señas específicas para trámites de ventanilla. Trabajamos con 3 usuarios para conseguir estas muestras. El modelo BiLSTM alcanzó una precisión del 92% con latencias inferiores a 150ms, cumpliendo así los requisitos para comunicación natural en tiempo real.

La evaluación exploratoria la hicimos con 6 usuarios oyentes y mostró alta percepción de utilidad institucional (4.5/5.0) junto con buena disposición para recomendar el sistema (83%). Identificamos como fortalezas la claridad de interfaz y el potencial social, mientras que las áreas donde se puede mejorar incluyen optimizar velocidad y precisión.

Las principales limitaciones que tenemos que reconocer son el dataset reducido con solo 3 usuarios - insuficiente para generalización robusta según estándares internacionales - y también que evaluamos únicamente con usuarios oyentes.

Palabras clave

Lenguaje de Señas Mexicano: La lengua natural que usa la comunidad sorda en México, reconocida oficialmente como lengua nacional. Tiene gramática y sintaxis propias que son diferentes del español oral - usa movimientos de manos, expresiones faciales y postura corporal para comunicarse.

Reconocimiento Gestual: Cuando las máquinas aprenden a leer y comprender gestos, sobre todo de las manos y brazos. En lenguas de señas, esto significa convertir movimientos a textos o voces.

Visión por Computadoras: Área de la inteligencia artificial que muestra a las máquinas cómo "ver" y analizar la información visual del mundo real por medio de cámaras. Es importante para que las computadoras estudien los movimientos gestuales.

Inteligencia Artificial: Tecnología que hace que las máquinas imiten actos de pensar humanos como aprender, razonar y elegir cosas. Aquí se usa para entender señas solas.

Redes Neuronales BiLSTM: Clase especial de inteligencia artificial muy buena para manejar secuencias de números que cambian con el tiempo. Lo "bidireccional" quiere decir que mira información hacia adelante y hacia atrás - ideal para comprender movimientos completos.

MediaPipe: Marco que hizo Google para ayudar con el análisis de video en tiempo real. Muy útil para encontrar puntos importantes en el cuerpo humano como manos, cara y forma.

Tiempo Real: Cuando un sistema mira información y da respuestas rápidamente, sin esperas que notes. Clave para hablar con claridad entre personas.

Comunicación Inclusiva: Manera que trata de quitar muros comunicativos para que todas las personas tomen parte en interacciones sociales sin importar lo que puedan o no hacer.

Discapacidad Auditiva: Condición que afecta la capacidad de oír sonidos un poco o por completo. Las personas con esta condición a menudo crecen habilidades visuales para comunicarse, como el lenguaje de manos.

Segmentación Automática: Cuando el ordenador sabe por si mismo cuándo comienza y cesa cada seña única dentro de un flujo seguido de movimientos. Quita necesidad de ayuda manual.

Clasificar Señas: Acto donde el sistema entiende qué seña concreta está haciendo dentro un grupo conocido. Es como reconocer formas en movimientos.

Aprendizaje Profundo: Área de inteligencia artificial que emplea redes neuronales con varias capas para captar formas complicadas en datos. Muy bueno para ver fotos y secuencias.

TensorFlow: Biblioteca de software libre que hizo Google para usar algoritmos en machine learning y deep learning. Es una de las herramientas más famosas para hacer sistemas inteligentes.

PyQt6: Estructura para crear formularios gráficos de usuario en Python. Deja hacer aplicaciones en ordenadores con ventanas, botones y cosas visuales buenas.

Sistema Múltiple: Sistema que trabaja y junta varios tipos de info como video, sonido y letras al mismo tiempo. Aquí maneja señales visuales y voz hablada.

Texto a Voz: Técnica que cambia letras en voz sintética que se puede oír. Ayuda a las máquinas a "hablar" el contenido textual.

Voz a Letras: Técnica opuesta que cambia sonido hablado a letras escritas. Deja que las máquinas "entiendan" lo que dice alguien.

Puntos Claves: Partes claras del cuerpo humano que se ven y siguen en videos, como articulaciones de los dedos, muñecas y codos. Esencial para entender gestos.

Dataset: Grupo de datos organizados usados para enseñar a algoritmos de mente. Aquí tiene grabaciones de señas con sus significados.

App de Escritorio: Programa que se pone en una computadora para funcionar, diferente a apps web que corren en navegadores.

Accesibilidad: Regla de diseño que quiere que productos y servicios se usen bien por gente con varias habilidades y limitaciones. Va más allá de seguir la ley hacia inclusión real.

Inclusión Social: Acto que verifica que todos tengan mismas chances de jugar en vida social, económica y cultural de su pueblo.

Personas Sordas: Individuos con pérdida auditiva fuerte que muchas veces creen una cultura propia y usan lenguas de señas como forma principal de hablar.

Trámites de Ventanilla: Pasos que se hacen en oficinas del gobierno, bancos, hospitales y otros lugares de servicio público, donde hay contacto directo.

Tecnología de Ayuda: Cualquier herramienta, programa o aparato que asiste a personas con discapacidades para hacer labores que podrían ser duras o imposibles si no tuvieran esas ayudas.

Procesamiento de Audio: Cambiar sonidos con un computador para hacer mejor la calidad, sacar detalles importantes o pasar entre varios tipos como voz y letra.

Interfaz de Usuario: El total de cosas para ver y tocar que dejan a los usuarios usar un sistema de manera fácil y rápida.

Arquitectura Modular: Método para hacer programas donde el sistema se parte en partes sueltas que se pueden crear, probar y cuidar por sí solas.

Evaluación de Usabilidad: Forma de ver qué tan fácil y bien se usa un sistema del punto de vista del usuario, hallando problemas y partes que pueden mejorar.

Culiacán Sinaloa: Es la ciudad principal del estado de Sinaloa, México, con alrededor de 915,000 personas. Lugar exacto donde se instaló y probó el sistema creado.

Índice general

1.	Introducción	1
1.1.	Contexto Demográfico de Culiacán	2
1.2.	Planteamiento del problema	2
1.2.1.	¿Dónde se manifiesta el problema?	3
1.2.2.	Impacto del problema	3
1.2.3.	Costos Económicos del Problema	4
1.2.4.	Limitaciones del Proyecto	5
1.2.5.	Marco Legal y Normativo	6
1.2.6.	Definición del Alcance: Vocabulario Específico para Trámites de Ventanilla 7	
1.3.	Objetivos.....	10
1.3.1.	Objetivo General.....	10
1.3.2.	Objetivos Específicos	10
1.4.	Justificación	10
1.4.1.	Viabilidad Técnica.....	11
1.4.2.	Impacto Proyectado	12
1.4.3.	Innovación y Contribución Académica	12
2.	Marco teórico	14
2.1.	Introducción al Marco Teórico	14
2.2.	Discapacidad Auditiva y Comunicación Inclusiva.....	14
2.2.1.	Modelos Conceptuales de la Discapacidad	15
2.2.2.	Principios de Comunicación Inclusiva	15
2.2.3.	Métodos Tradicionales de Comunicación Inclusiva.....	15
2.2.4.	Limitaciones de los Enfoques Tradicionales	16
2.3.	Lenguaje de Señas Mexicano (LSM)	16
2.3.1.	Estructura Lingüística del LSM.....	16
2.3.2.	Características Sintácticas Distintivas	17
2.3.3.	Variabilidad Diatópica y Diastrática	18
2.3.4.	LSM en Contextos Institucionales.....	18
2.3.5.	Fundamentos de Visión por Computadora para Reconocimiento Gestual ...	19
2.3.6.	Pipeline Conceptual de Procesamiento Visual	19

2.3.7.	Técnicas de Representación de Características	19
2.3.8.	Desafíos Técnicos en Reconocimiento Gestual.....	20
2.3.9.	Métricas de Evaluación en Visión por Computadora	21
2.4.	Procesamiento de Señales de Audio para Sistemas Multimodales	21
2.4.1.	Fundamentos de Digitalización de Audio	21
2.4.2.	Técnicas de Segmentación de Voz	22
2.4.3.	Preprocesamiento de Señales de Audio.....	23
2.4.4.	Calidad de Audio para Reconocimiento Automático	23
2.5.	Síntesis de Voz y Text-to-Speech.....	24
2.5.1.	Fundamentos de Síntesis de Voz.....	24
2.5.2.	Tipos de Síntesis de Voz	24
2.5.3.	Métricas de Calidad en TTS	25
2.5.4.	Consideraciones para Aplicaciones de Accesibilidad	25
2.6.	Inteligencia Artificial para Reconocimiento de Secuencias Temporales	26
2.6.1.	Evolución de Arquitecturas para Secuencias.....	26
2.6.2.	Redes Neuronales Recurrentes (RNN)	26
2.6.3.	Bidirectional LSTM (BiLSTM)	27
2.6.4.	Técnicas de Regularización y Optimización	28
2.6.5.	Métricas de Evaluación para Clasificación Temporal.....	28
2.7.	Arquitecturas de Software para Sistemas Multimodales.....	29
2.7.1.	Patrones de Diseño para Aplicaciones de Tiempo Real.....	29
2.7.2.	Arquitecturas Cliente-Servidor vs Standalone	30
2.7.3.	Frameworks de Desarrollo para Aplicaciones Desktop	30
2.7.4.	Gestión de Concurrencia en Aplicaciones Multimedia	30
2.8.	Sistemas de Tiempo Real	31
2.8.1.	Clasificación de Sistemas de Tiempo Real	31
2.8.2.	Técnicas de Optimización de Rendimiento	31
2.8.3.	Gestión de Recursos en Sistemas Multimedia.....	32
2.9.	Frameworks y Bibliotecas para Desarrollo de Sistemas LSM	32
2.9.1.	Criterios de Selección de Tecnologías	32
2.9.2.	Comparativa de Bibliotecas de Visión por Computadora	32
2.9.3.	Frameworks de Deep Learning.....	33

2.9.4.	Frameworks de Desarrollo de Interfaces	33
2.10.	Conclusiones del Marco Teórico.....	34
3.	Estado del arte	35
3.1.	Introducción al estado del arte.....	35
3.1.1.	Alcance y Criterios de Revisión	35
3.2.	Sistemas de Reconocimiento Automático de Lenguas de Señas.....	36
3.2.1.	Aproximaciones Basadas en Extracción de Características	36
3.2.2.	Sistemas Híbridos con Múltiples Modalidades	36
3.2.3.	Análisis de Rendimiento de Sistemas Existentes	37
3.3.	Arquitecturas Recurrentes Aplicadas a Reconocimiento Gestual	37
3.3.1.	Fundamentos de LSTM en Reconocimiento de Secuencias.....	37
3.3.2.	Bidirectional LSTM para Reconocimiento Gestual	37
3.3.3.	Comparación LSTM vs BiLSTM en Aplicaciones Gestuales.....	38
3.3.4.	Optimizaciones Específicas para Tiempo Real	38
3.4.	Técnicas de Segmentación Automática de Gestos	38
3.4.1.	Métodos Basados en Umbrales de Movimiento	39
3.4.2.	Modelos Estadísticos para Segmentación.....	39
3.4.3.	Redes Neuronales para Segmentación Temporal	40
3.4.4.	Segmentación en Tiempo Real para Aplicaciones Desktop	40
3.5.	Sistemas Multimodales con Procesamiento de Audio.....	41
3.5.1.	Speech-to-Text en Tiempo Real	41
3.5.2.	Text-to-Speech para Retroalimentación Auditiva	42
3.6.	Aplicaciones de Escritorio para Comunicación Inclusiva.....	42
3.6.1.	Frameworks de Desarrollo para Aplicaciones Accesibles	42
3.6.2.	Arquitecturas de Aplicaciones Multimodales	43
3.7.	Datasets y Benchmarks de Lenguas de Señas	44
3.7.1.	Datasets Internacionales Principales	44
3.7.2.	Ausencia de Datasets Públicos de LSM	45
3.7.3.	Datasets de Investigación Mexicana.....	45
3.8.	Trabajos Específicos en Lenguaje de Señas Mexicano	46
3.8.1.	Investigación Académica Nacional	46
3.8.2.	Colaboraciones con Organizaciones de Personas Sordas	47

3.8.3.	Limitaciones de la Investigación Nacional.....	47
3.9.	Análisis Comparativo y Limitaciones de Trabajos Previos.....	47
3.9.1.	Sesgo Cultural y Lingüístico en la Investigación	47
3.9.2.	Limitaciones Técnicas Recurrentes	48
3.9.3.	Deficiencias en Evaluación de Usabilidad	48
3.9.4.	Limitaciones Arquitecturales.....	49
3.9.5.	Brechas en Sistemas de Tiempo Real.....	49
3.10.	Sistemas de Aplicaciones de Escritorio vs Web.....	50
3.10.1.	Ventajas de Aplicaciones Desktop para LSM	50
3.10.2.	Desafíos de Distribución y Mantenimiento	50
3.11.	Métricas de Evaluación y Benchmarking	51
3.11.1.	Métricas Técnicas Estándar	51
3.11.2.	Métricas de Usabilidad Específicas	52
3.12.	Tendencias Emergentes y Trabajo Futuro	52
3.12.1.	Modelos Multimodales Avanzados	52
3.12.2.	Adaptación a Usuarios Individuales	53
3.12.3.	Integración con Realidad Aumentada.....	53
3.13.	Análisis de Brechas y Oportunidades.....	54
3.13.1.	Brechas Identificadas en la Literatura	54
3.13.2.	Contribuciones Únicas del Presente Trabajo.....	54
3.13.3.	Limitaciones Reconocidas.....	54
3.14.	Conclusiones del Estado del Arte	55
4.	Desarrollo del proyecto	57
4.1.	Introducción al Desarrollo	57
4.1.1.	Metodología de Desarrollo Tecnológico Experimental.....	57
4.1.2.	Cronograma de Desarrollo.....	57
4.1.3.	Desafíos y Adaptaciones Durante el Desarrollo.....	58
4.1.4.	Herramientas y Entorno de Desarrollo	60
4.2.	Arquitectura General del Sistema.....	60
4.2.1.	Diseño Arquitectural.....	60
4.2.2.	Componentes Principales	61
4.2.3.	Flujo de Datos.....	62

4.3.	Desarrollo del Sistema de Autenticación.....	62
4.3.1.	Arquitectura Híbrida Desktop-Web.....	62
4.3.2.	Implementación de OAuth2.....	62
4.3.3.	Gestión de Base de Datos	63
4.4.	Desarrollo del Motor de Reconocimiento LSM	64
4.4.1.	Arquitectura del Sistema de Visión	64
4.4.2.	Implementación de MediaPipe	65
4.4.3.	Algoritmo de Segmentación Automática	66
4.4.4.	Estructura de Datos de Entrenamiento	67
4.5.	Desarrollo del Modelo BiLSTM	68
4.5.1.	Arquitectura del Modelo.....	68
4.5.2.	Preprocesamiento de Datos	69
4.5.3.	Entrenamiento y Optimización.....	70
4.5.4.	Resultados del Entrenamiento	71
4.5.5.	Optimización del Modelo	73
4.6.	Desarrollo del Sistema Multimodal de Audio	74
4.6.1.	Captura de Voz en Tiempo Real.....	74
4.6.2.	Integración con Google Cloud Speech-to-Text	75
4.6.3.	Síntesis de Voz Local	75
4.7.	Desarrollo de la Interfaz de Usuario (PyQt6).....	76
4.7.1.	Arquitectura de la Interfaz.....	76
4.7.2.	Sistema de Navegación Animada	76
4.7.3.	Widget de Traducción - Interfaz Principal	77
4.7.4.	Sistema de Configuración de Cámara	78
4.8.	Integración de Módulos y Optimizaciones	79
4.8.1.	Gestión de Hilos y Concurrencia.....	79
4.8.2.	Optimizaciones de Rendimiento.....	80
4.9.	Sistema de Suscripciones Premium.....	81
4.9.1.	Integración con Stripe.....	81
4.10.	Pruebas de Integración y Validación	83
4.10.1.	Metodología de Pruebas	83
4.10.2.	Métricas de Rendimiento Obtenidas	83

4.10.3.	Resolución de Problemas Críticos.....	85
4.11.	Resultados del Desarrollo.....	87
4.11.1.	Componentes Completados Según Cronograma	87
4.11.2.	Arquitectura Final Implementada.....	88
4.11.3.	Métricas de Rendimiento Final	89
4.11.4.	Contribuciones Técnicas Innovadoras.....	89
4.11.5.	Impacto y Escalabilidad.....	90
4.11.6.	Limitaciones Críticas del Desarrollo	91
4.11.7.	Dependencias Externas y Requisitos del Sistema	92
4.12.	Conclusiones del Desarrollo.....	94
5.	Pruebas y Resultados.....	96
5.1.	Introducción a la Evaluación	96
5.2.	Metodología de Evaluación.....	96
5.2.1.	Diseño del Estudio.....	96
5.2.2.	Participantes	97
5.2.3.	Instrumentos de Medición	97
5.2.4.	Procedimiento.....	98
5.3.	Resultados Cuantitativos	99
5.3.1.	Estadísticas Descriptivas	99
5.3.2.	Análisis de Índices Compuestos.....	99
5.3.3.	Análisis por Nivel de Experiencia Tecnológica	100
5.3.4.	Distribución de Respuestas por Variable.....	101
5.4.	Análisis Cualitativo	101
5.4.1.	Aspectos Más Valorados del Sistema.....	101
5.4.2.	Problemas Identificados	102
5.4.3.	Sugerencias de Mejora.....	103
5.4.4.	Análisis de Correlaciones	103
5.5.	Síntesis de Resultados	104
5.5.1.	Fortalezas Identificadas	104
5.5.2.	Áreas Críticas de Mejora	104
5.5.3.	Patrones de Evaluación.....	105
5.6.	Limitaciones del Estudio	105

5.6.1.	Limitaciones Metodológicas	105
5.6.2.	Limitaciones de Validez	106
5.6.3.	Implicaciones de las Limitaciones.....	106
5.7.	Conclusiones del Capítulo	107
5.7.1.	Hallazgos Principales	107
5.7.2.	Validación de Objetivos de Evaluación.....	107
5.7.3.	Implicaciones para Desarrollo Futuro	108
6.	Conclusiones y trabajo futuro	109
6.1.	Introducción.....	109
6.2.	Cumplimiento de Objetivos.....	109
6.2.1.	Evaluación del Objetivo General.....	109
6.2.2.	Evaluación de Objetivos Específicos	110
6.3.	Respuesta al Planteamiento del Problema	111
6.3.1.	Problema Original Identificado	111
6.3.2.	Solución Desarrollada.....	112
6.3.3.	Impacto Proyectado	112
6.4.	Contribuciones Principales	113
6.4.1.	Contribuciones Técnicas.....	113
6.4.2.	Contribuciones Metodológicas	113
6.4.3.	Contribuciones Sociales	114
6.5.	Limitaciones y Restricciones.....	114
6.5.1.	Limitaciones Metodológicas Críticas	114
6.5.2.	Limitaciones Técnicas	115
6.5.3.	Limitaciones de Escalabilidad	115
6.6.	Implicaciones Prácticas	115
6.6.1.	Para Instituciones Públicas y Privadas	115
6.6.2.	Para la Comunidad Académica	116
6.6.3.	Para Políticas Públicas.....	116
6.7.	Trabajo Futuro	116
6.7.1.	Prioridades de Investigación a Corto Plazo (1-2 años).....	116
6.8.	Recomendaciones Específicas	117
6.8.1.	Para Investigadores Futuros.....	117

6.8.2.	Para Instituciones Implementadoras	117
6.8.3.	Para Formuladores de Políticas	118
6.9.	Reflexiones Finales	118
6.9.1.	Sobre el Proceso de Investigación	118
6.9.2.	Sobre el Impacto Social de la Tecnología	118
6.9.3.	Sobre el Estado de la Investigación en LSM.....	119
6.9.4.	Sobre la Viabilidad de Soluciones Tecnológicas	119
6.10.	Conclusión General	119
7.	Bibliografía.....	121
8.	Anexos.....	127
8.1.	Anexo A: INSTRUMENTOS DE EVALUACIÓN Y RESULTADOS	127
8.2.	ANEXO B: REGISTRO VISUAL DE LAS EVALUACIONES	129
8.3.	Anexo C: Código Fuente del Sistema	8.3-1
8.4.	Anexo D: Dataset de Entrenamiento	8.4-1

Índice de figuras

Figura 2-1: Parámetros Formativos del LSM.....	17
Figura 2-2: Mapa Conceptual de Variación en LSM	18
Figura 2-3: Pipeline General de Procesamiento Visual.....	19
Figura 2-4: Proceso de Digitalización de Audio	21
Figura 2-5: Arquitectura Conceptual de Sistema TTS	24
Figura 2-6: Evolución de Arquitecturas para Secuencias Temporales.....	26
Figura 2-7: Arquitectura Conceptual de Neurona LSTM.....	27
Figura 2-8: Patrones Arquitecturales para Sistemas Multimodales.....	29
Figura 4-1: Arquitectura en Capas de la Aplicación de Reconocimiento de Lenguaje de Señas	61
Figura 4-2: Diagrama de Flujo del Sistema Propuesto de Traducción Multimodal	62
Figura 4-3: Curvas de Precisión y Pérdida del Modelo Durante el Entrenamiento	71
Figura 4-4: Matriz de Confusión del Modelo de Clasificación de Lenguaje de Señas	72
Figura 4-5: Métricas de Rendimiento Detalladas por Clase de Lenguaje de Señas	72
Figura 4-6: métricas del rendimiento en cuanto latencias obtenidas en ejecución.....	83
Figura 4-7: Monitoreo de Actividad de CPU Durante el Entrenamiento	84
Figura 4-8: Rendimiento de Precisión por Tipo de Señal.....	84
Figura 4-9: Arquitectura del Sistema de Aplicación PyQtG Nativa	89

Índice de tablas

Tabla 2-1: Comparación de Modelos de Discapacidad	14
Tabla 2-2: Comparación Sintáctica LSM vs Español.....	17
Tabla 2-3: Comparativa de Técnicas de Extracción de Características.....	20
Tabla 2-4: Comparativa de Métodos VAD.....	22
Tabla 2-5: Comparativa de Tecnologías TTS.....	25
Tabla 2-6: Comparativa de Arquitecturas Recurrentes	27
Tabla 2-7: Comparativa Arquitectural para Sistemas LSM	30
Tabla 2-8: Comparativa de Frameworks Desktop	30
Tabla 2-10: Requisitos Temporales para Comunicación Humana	31
Tabla 2-11: Frameworks de Detección de Pose/Keypoints.....	32
Tabla 2-12: Comparativa de Frameworks de ML	33
Tabla 3-1. Comparativa de Sistemas de Reconocimiento de Señas Documentados	37
Tabla 3-2. Rendimiento Comparativo LSTM vs BiLSTM	38
Tabla 3-3. Comparativa de motores TTS	42
Tabla 3-4. Comparativa de Frameworks para Aplicaciones Accesibles	43
Tabla 3-5. Distribución de Estudios por Lengua de Señas.....	48
Tabla 3-6. Requerimientos de Hardware en Sistemas Revisados	49
Tabla 3-7. Comparativa Desktop vs Web para Reconocimiento LSM	50
Tabla 3-8. Síntesis de Brechas en el Estado del Arte	54
Tabla 4-1: Métricas de Robustez bajo Condiciones de Prueba	84
Tabla 4-2: Análisis de Falsos Positivos y Negativos del Sistema	85
Tabla 4-3: Comparación con Estándares y Conjuntos de Datos Internacionales	91
Tabla 5-1: Estadísticas Descriptivas de Variables de Evaluación	99
Tabla 5-2: índices Compuestos de Evaluación.....	99
Tabla 5-3: Resultados por Nivel de Experiencia Tecnológica	100
Tabla 5-4: Distribución de Frecuencias de Respuestas (n=6)	101
Tabla 5-5: Aspectos Más Valorados del Sistema	101
Tabla 5-6: Problemas Principales Identificados	102
Tabla 5-7: Sugerencias de Mejora Categorizadas	103
Tabla 5-8: Matriz de Correlaciones entre Variables Clave	103

Capítulo 1

1.Introducción

En este trabajo de tesis se documenta la investigación, el diseño y también la implementación de un sistema de inteligencia artificial que sirve para interpretar el Lenguaje de Señas Mexicano (LSM) en tiempo real. El desarrollo se orientó específicamente a facilitar trámites en ventanilla y mejorar cómo se comunican las personas sordas con el personal oyente en instituciones de Culiacán. Esta iniciativa nace de la necesidad de superar las barreras comunicativas que se presentan en trámites diarios, sobre todo en entornos tan cruciales como oficinas de gobierno, bancos, hospitales y centros de atención.

A pesar de que existen avances normativos y leyes específicas como la Ley General para la Inclusión de las Personas con Discapacidad (2011), en la práctica diaria las herramientas tecnológicas efectivas siguen siendo escasas, y esto limita significativamente la accesibilidad real. Este proyecto propone, por tanto, una solución innovadora que combina tecnologías como MediaPipe para detectar de manera precisa los movimientos gestuales y redes neuronales bidireccionales de memoria a largo y corto plazo (BiLSTM), para traducir con precisión gestos capturados en texto de forma fluida.

Se realizó un análisis exhaustivo sobre las tecnologías disponibles para poder obtener los mejores resultados en el menor tiempo posible.

Este análisis previo fue el que fundamentó la selección estratégica de MediaPipe y BiLSTM, herramientas que destacaron sobre otras alternativas por su demostrada robustez, alta precisión y eficacia en aplicaciones análogas de visión por computadora. Una vez definida la base tecnológica, el siguiente paso crucial fue el diseño de un algoritmo de segmentación. Este algoritmo no fue una tarea nada fácil, tal vez fue lo más difícil del desarrollo y su propósito es: identificar de forma automática el instante exacto en que una señal comienza y termina dentro de un flujo de movimiento continuo, asegurando así que cada gesto se aísle correctamente para su posterior clasificación a través de la red neuronal ya entrenada.

Las pruebas del sistema se efectuaron en ambientes controlados (se esperaba que fueran usuarios sordos de ser posible), pero tras realizar encuestas al personal administrativo del

Tecnológico de Culiacán, se concluyó que las condiciones no eran óptimas para una validación en contextos reales debido a la ausencia actual de usuarios con estas características en dichos entornos.

Entre las principales limitaciones del proyecto destacan la variabilidad del Lenguaje de Señas Mexicano, debido a diferencias regionales y contextuales significativas, así como los desafíos técnicos relacionados con la precisión del reconocimiento en condiciones ambientales variables (iluminación irregular, movimiento de fondo, etc.). En base a las limitaciones se priorizó la relevancia sobre la amplitud, lo que resultó en un conjunto de 28 palabras y frases seleccionadas por ser las que representan las interacciones más frecuentes y funcionales en ventanillas y otros espacios de atención al público.

1.1. Contexto Demográfico de Culiacán

La investigación se orientó a Culiacán, capital de Sinaloa, que cuenta con una población de aproximadamente 915,000 habitantes según el Censo de Población y Vivienda 2020 del Instituto Nacional de Estadística y Geografía (INEGI). De esta población, se estima que aproximadamente 6,771 personas presentan algún grado de discapacidad auditiva. Se llegó a esta cifra aplicando la proporción nacional del 0.74% reportada por INEGI (2020) a la población municipal de Culiacán (915,000 habitantes), esto ya que no se cuenta con estudios locales.

Culiacán cuenta con aproximadamente 45 oficinas gubernamentales de nivel municipal y estatal, 120 sucursales bancarias de diferentes instituciones financieras, 15 hospitales públicos y 8 centros de salud privados principales. Esto demuestra que hay muchos lugares vulnerables donde se podrían presentar estas barreras comunicativas, que afectan tanto a las personas con discapacidad auditiva.

1.2. Planteamiento del problema

En Culiacán, las personas sordas o con discapacidad auditiva enfrentan barreras significativas que dificultan su comunicación efectiva con el resto de la población oyente, particularmente al realizar trámites esenciales en ventanillas de atención al público. Esta situación limita

severamente su acceso equitativo a servicios fundamentales como atención médica, trámites gubernamentales, operaciones bancarias y servicios comerciales, además de restringir notablemente su participación en actividades laborales y sociales. Aunque México cuenta con leyes que promueven la inclusión y accesibilidad, como la Ley General para la Inclusión de las Personas con Discapacidad (2011), en la práctica diaria estas normativas no se reflejan de manera efectiva debido a la ausencia o escasez de herramientas tecnológicas prácticas y accesibles que faciliten la comunicación inclusiva.

1.2.1. ¿Dónde se manifiesta el problema?

Esta problemática resulta bastante notoria en los lugares donde se atiende al público en Culiacán. Hablamos de oficinas del gobierno municipal y estatal, bancos, hospitales públicos y privados, centros comerciales, y también en eventos comunitarios y culturales que se organizan en la ciudad. En todos estos espacios, al no existir soluciones tecnológicas pensadas específicamente para el Lenguaje de Señas Mexicano (LSM), surgen complicaciones importantes a la hora de dar atención, realizar trámites básicos y establecer comunicación diaria. La falta de estas herramientas no únicamente afecta a las personas sordas, también limita que instituciones y negocios logren ofrecer servicios verdaderamente inclusivos y de calidad, impactando su responsabilidad social y la manera en que la gente ve su desempeño.

Nota Metodológica sobre Datos Demográficos: Los datos que utilizamos provienen del Censo de Población y Vivienda 2020 del INEGI, que contabilizó 948,000 personas con discapacidad auditiva a nivel nacional (0.74% de la población total). Para Culiacán aplicamos esta proporción nacional porque no hay cifras municipales específicas disponibles. Hay que considerar que estos números pueden cambiar según los criterios usados para clasificar la discapacidad auditiva (parcial contra total, congénita contra adquirida).

1.2.2. Impacto del problema

Esta situación tiene un impacto bastante amplio y toca varios sectores de manera directa:

Instituciones y negocios en Culiacán: Cuando las instituciones no tienen herramientas para atender personas sordas, pues obviamente se compromete la calidad del servicio que ofrecen. Pero además de eso, también se puede ver afectada su reputación, los usuarios pierden confianza y hasta pueden meterse en problemas legales por no cumplir las normas de

accesibilidad que existen. Se ha documentado que cuando hay estas barreras de comunicación, el tiempo que se tarda en atender aumenta como un 40% en promedio, y esto obviamente afecta qué tan eficiente puede ser la operación en general.

La sociedad de Culiacán: Al no contar con tecnologías que sean inclusivas, se siguen manteniendo las barreras tanto sociales como económicas, y esto hace más complicado que las personas sordas puedan integrarse bien a las actividades de todos los días. Reduce las oportunidades que tienen estos ciudadanos para participar de forma activa en la vida económica, social y cultural de nuestra ciudad, lo cual limita que podamos avanzar hacia una comunidad que sea más justa e igualitaria.

Personas sordas de Culiacán: Las barreras para comunicarse generan exclusión social, bajan mucho su calidad de vida y les limita el acceso igualitario a oportunidades educativas y de trabajo. Estas dificultades que enfrentan constantemente afectan directamente qué tan bien pueden expresar sus necesidades básicas, acceder a servicios que son esenciales y desarrollarse como personas de manera completa en un entorno social que no está bien adaptado a lo que necesitan específicamente.

1.2.3. Costos Económicos del Problema

Cuando no se tienen herramientas tecnológicas para comunicación inclusiva, se generan costos económicos que ya están documentados. Estos costos afectan tanto a las personas sordas como a las instituciones, y son bastante considerables.

Costos que las personas sordas tienen que enfrentar: Contratar intérpretes profesionales de LSM es bastante caro. Las tarifas van desde \$300 hasta \$600 pesos mexicanos por hora. Estos datos los sacamos del directorio de intérpretes certificados que maneja el DIF Sinaloa (2023). Directorio de Intérpretes Certificados de Lenguaje de Señas Mexicano. Sistema para el Desarrollo Integral de la Familia del Estado de Sinaloa. Cuando se trata de trámites complicados que pueden durar 2 o 3 horas, esto representa un costo que va de \$600 a \$1,800 pesos por cada gestión. Y esto es algo que las personas tienen que pagar de su bolsillo.

Costos que tienen que absorber las instituciones: Las instituciones se topan con incrementos en los tiempos de atención de aproximadamente 40% en promedio cuando atienden a personas sordas y no cuentan con las herramientas adecuadas. Esto lo pudimos

observar nosotros mismos en oficinas gubernamentales locales durante la fase de investigación de este proyecto. Esto se traduce en costos operacionales adicionales porque se alargan los tiempos que se necesitan para dar el servicio, y eso cuesta dinero.

Nota metodológica: Para poder cuantificar de manera precisa cuáles son los impactos económicos reales se necesitarían estudios longitudinales especializados. Esto está fuera del alcance que tiene este proyecto de desarrollo tecnológico, pero sería algo interesante de investigar a futuro.

1.2.4. Limitaciones del Proyecto

Este trabajo tiene limitaciones metodológicas importantes que afectan qué tanto se pueden generalizar los resultados:

LIMITACIÓN CRÍTICA - Dataset Insuficiente para Generalización:

- El dataset se limitó a 3 usuarios, lo cual está muy por debajo de lo que requieren los estándares académicos internacionales (mínimo 20-30 usuarios para ML que sea robusto)
- Esta limitación es la principal debilidad metodológica que tiene el estudio
- Los resultados que obtuvimos (92% de precisión) se deben interpretar como una prueba de concepto técnico, NO como validación de un sistema que esté listo para implementarse
- Se necesita investigación futura con datasets más amplios (≥ 20 usuarios) para poder confirmar la robustez y generalización

Limitaciones Adicionales:

- Vocabulario experimental de 28 señas que no tiene validación institucional en contextos reales
- La evaluación se limitó a condiciones controladas, sin implementación en ventanillas reales
- Hay sesgo hacia la variante LSM de Culiacán/Sinaloa, sin representar la diversidad regional
- Dependencia de servicios en la nube (Google Speech API) para que funcione completamente

- Las restricciones de tiempo de 16 semanas no permitieron recolectar un dataset robusto
- Dependencia crítica de servicios externos: Google Cloud Speech API para convertir voz a texto, lo cual limita la funcionalidad cuando no hay conectividad

Implicaciones: Los resultados que se obtuvieron (92% precisión) deben interpretarse como prueba de concepto técnico, no como validación de un sistema listo para implementación masiva. Se requiere investigación adicional con datasets ampliados y evaluación en contextos reales para confirmar que el sistema sea viable en la práctica.

1.2.5. Marco Legal y Normativo

El problema de comunicación que viven las personas sordas en Culiacán se ubica dentro de un contexto legal que establece de manera bastante clara la obligación de garantizar accesibilidad y no discriminación. Sin embargo, en la práctica su implementación sigue siendo deficiente, por decir lo menos.

Marco internacional: México es firmante de la Convención sobre los Derechos de las Personas con Discapacidad de las Naciones Unidas (2006), que se ratificó en 2007. En su Artículo 21 establece el derecho a la libertad de expresión y acceso a la información, y esto incluye que se reconozca y promueva el uso de lenguas de señas. También tenemos el Convenio 159 de la Organización Internacional del Trabajo sobre la readaptación profesional y el empleo de personas inválidas, que establece principios de igualdad de oportunidades laborales. Estas son obligaciones internacionales que México tiene.

Marco nacional: La Constitución Política de los Estados Unidos Mexicanos, en su Artículo 1º, prohíbe cualquier tipo de discriminación que esté motivada por discapacidad. La Ley General para la Inclusión de las Personas con Discapacidad (2011) establece en su Artículo 14 la obligación de garantizar el acceso a la información y comunicación, y esto incluye que se reconozca el LSM como lengua nacional. La Ley Federal para Prevenir y Eliminar la Discriminación (2003) complementa este marco legal y establece medidas positivas y compensatorias para personas con discapacidad. En papel, todo se ve muy bien.

Marco técnico: La NOM-034-STPS-2016 sobre condiciones de seguridad para el acceso y desarrollo de actividades de trabajadores con discapacidad establece criterios específicos de

accesibilidad que incluyen aspectos comunicacionales. Aunque esta norma se enfoca principalmente en espacios laborales, sienta precedentes importantes para implementar herramientas tecnológicas de comunicación inclusiva en otros espacios.

Cumplimiento y brechas: A pesar de tener este marco legal bastante robusto, existe una brecha muy significativa entre la normatividad y cómo se implementa en la realidad. El Consejo Nacional para el Desarrollo y la Inclusión de las Personas con Discapacidad (CONADIS) reporta que menos del 30% de las instituciones públicas cumplen completamente con los criterios de accesibilidad comunicacional. Esto evidencia la necesidad urgente de soluciones tecnológicas como la que se propone en esta tesis, porque las leyes solas no bastan.

1.2.6. Definición del Alcance: Vocabulario Específico para Trámites de Ventanilla

Este proyecto se enfoca de manera deliberada en un vocabulario limitado, pero que fue seleccionado estratégicamente de 28 señas del LSM. Esta decisión responde a criterios técnicos, metodológicos y de viabilidad práctica. Se fundamenta en las siguientes consideraciones que explicamos a continuación:

Criterios de Selección del Vocabulario:

1. Validación de la Problemática

La investigación preliminar incluyó entrevistas semi-estructuradas con personal de instituciones locales. El objetivo era validar que existe el problema y qué tan grande es la brecha comunicativa entre personas sordas y servicios presenciales. Los resultados se organizaron por sector, y fueron bastante reveladores:

Institución educativa: Tecnológico de Culiacán

- Participantes: 3 administrativos del área de ventanilla de control escolar (2 mujeres, 1 hombre).
- Modalidad: Consultas exploratorias informales.
- Percepciones clave:
 - La implementación de herramientas de comunicación inclusiva sería "excelente y ayudaría mucho".

- Se identificó una paradoja que se pudo observar: muy pocos casos visibles de personas sordas realizando trámites.
- Hipótesis del personal: las personas sordas son "reacias a ir a hacer trámites" debido a las barreras comunicativas.
- Implicación: Existe una demanda latente significativa, que actualmente no está satisfecha por la ausencia de herramientas adecuadas. Es un problema real pero invisible.

Sector comercial: Woolworth (tienda departamental)

- Participante: 1 cajera con experiencia directa en atención a clientes sordos.
- Modalidad: Entrevista informal.
- Hallazgos:
 - Frecuencia estimada: aproximadamente 3 casos de personas sordas por mes, por cajera.
 - Se confirmó la dificultad de comunicación efectiva, citando experiencias personales.
 - Respuesta proactiva del personal: algunos colaboradores han comenzado a aprender LSM por iniciativa propia, lo cual es admirable.
- Implicación: Existe una necesidad real de herramientas comunicativas, especialmente en entornos con exposición regular a usuarios sordos.

Conclusiones de la validación La investigación preliminar permitió confirmar tanto:

1. La existencia del problema: dificultades comunicativas reales entre personas sordas y personal de atención.
2. Su efecto disuasorio: la baja frecuencia de personas sordas realizando trámites puede ser síntoma de exclusión por barreras de acceso.

En conjunto, los hallazgos validan qué tan relevante es el proyecto y destacan la necesidad de soluciones tecnológicas inclusivas que permitan reducir estas barreras en contextos institucionales y comerciales.

2. Limitaciones Técnicas del Prototipo Como sistema experimental desarrollado en entorno académico, el alcance se limitó intencionalmente para garantizar: • Precisión alta (>90%) en el vocabulario seleccionado vs precisión mediocre (60-70%) en vocabulario extenso • Tiempo de entrenamiento viable (8 horas vs semanas para vocabularios extensos) • Dataset manejable con recursos disponibles (1,400 muestras vs 50,000+ requeridas para vocabularios grandes) • Evaluación exhaustiva de cada señal vs evaluación superficial de muchas señales

3. Enfoque en Comunicación Esencial Tomando como base la retroalimentación del personal entrevistado y la literatura especializada en comunicación institucional, las 28 señales seleccionadas cubren las interacciones más comunes que se identificaron como necesarias en trámites de ventanilla: • Saludos y cortesía (6 señales): Hola, Buenos días, Por favor, Gracias, Disculpe, Adiós • Información personal (8 señales): Mi nombre es, Apellido, Edad, Dirección, Teléfono, Email, Firma, Identificación • Términos de trámites (8 señales): Documento, Dinero, Pagar, Cuánto, Sí, No, Ayuda, Problema • Alfabeto manual básico (6 señales): A, E, I, O, U, V

4. Precedentes en Literatura Académica La investigación en reconocimiento automático de lenguas de señas frecuentemente utiliza vocabularios limitados en fases iniciales. Esto no es algo nuevo: • Sistemas ASL: 50-100 palabras en prototipos académicos • Sistemas BSL: 30-80 palabras en implementaciones experimentales • Sistemas CSL: 40-120 palabras en estudios de viabilidad

Justificación Metodológica: Un vocabulario reducido, pero que esté bien implementado genera mayor impacto práctico que un vocabulario extenso con baja precisión. Este enfoque permite: • Validación rigurosa de la arquitectura técnica que se propone • Evaluación detallada de usabilidad con usuarios reales • Identificación de patrones de error para mejoras futuras • Establecimiento de línea base para expansiones posteriores

1.3. Objetivos

1.3.1. Objetivo General

Desarrollar un prototipo de software que funcione como intérprete de lenguaje de señas en tiempo real. Esto como una prueba de concepto técnico que demuestre que sí es viable tener comunicación inclusiva LSM-español a través de pruebas controladas.

1.3.2. Objetivos Específicos

- Identificar y documentar cuáles son los requerimientos técnicos y funcionales que necesita el sistema. Esto mediante análisis de contextos reales de atención en ventanilla, y definiendo un vocabulario específico que sea de alta frecuencia en los trámites cotidianos.
- Diseñar una interfaz de usuario que sea funcional, accesible e intuitiva, para que cualquier persona pueda utilizar el sistema desde un equipo de escritorio en ambientes que estén controlados o simulados.
- Desarrollar e integrar un algoritmo que detecte e interprete el Lenguaje de Señas Mexicano (LSM), y que sea capaz de traducir gestos en tiempo real tanto a texto como a voz.
- Hacer pruebas funcionales del software con usuarios sordos y oyentes en entornos controlados, esto para validar qué tan preciso y eficiente es el sistema sin tener que depender de despliegues institucionales.
- Documentar cuáles fueron los resultados que se obtuvieron y generar una propuesta para implementación futura.

1.4. Justificación

En Culiacán, las personas sordas siguen enfrentándose a barreras comunicativas bastante importantes en situaciones cotidianas, sobre todo cuando necesitan realizar trámites en ventanilla. Esta falta de accesibilidad no solamente limita su capacidad para interactuar de manera autónoma y efectiva, también les restringe el acceso a servicios esenciales y su participación plena en la vida social y económica. Aunque ya existen leyes que respaldan el derecho a la inclusión, como la Ley General para la Inclusión de las Personas con

Discapacidad y la Convención sobre los Derechos de las Personas con Discapacidad de la ONU, pues la ausencia de soluciones tecnológicas prácticas ha impedido que estas normativas se traduzcan en mejoras que sean tangibles en la vida diaria.

Ante esta situación, desarrollar un sistema que permita interpretar el Lenguaje de Señas Mexicano (LSM) en tiempo real representa una respuesta concreta y que sí es factible al problema. La propuesta consiste en un software que traduce señas a texto y voz usando algoritmos de inteligencia artificial que están basados en redes neuronales bidireccionales LSTM. El objetivo es facilitar la comunicación entre personas sordas y oyentes en trámites. La aplicación está pensada para ser utilizada con bajos recursos y en casi cualquier espacio de ventanilla, sin necesidad de infraestructura compleja.

1.4.1. Viabilidad Técnica

Hablando del aspecto tecnológico, este proyecto se basa en herramientas que están disponibles en la actualidad, como MediaPipe para poder extraer keypoints y redes neuronales BiLSTM que se pueden implementar usando TensorFlow. Estas ya han probado que son eficaces cuando se usan en aplicaciones de visión por computadora y para procesar secuencias temporales. Decidimos usar BiLSTM en vez de arquitecturas que son más complejas como los Transformers, porque tiene un buen equilibrio entre precisión y eficiencia computacional, cosa que permite que funcione en tiempo real usando hardware estándar.

Requerimientos de hardware: Diseñamos el sistema para que pueda operar con equipamiento accesible: una cámara web HD 1080p (que cuesta más o menos \$1000.00 MX), computadora de escritorio que tenga procesador de gama media y unos 8GB de RAM. Con esta configuración logramos mantener latencias que estén por debajo de los 200ms, y esto es lo que se considera como el umbral para que la percepción sea natural cuando hay comunicación humana.

Precisión demostrada: En las pruebas preliminares que realizamos con el modelo BiLSTM que desarrollamos, vimos una precisión general del 92%. Pero esto varía dependiendo de qué tan compleja sea cada seña: conseguimos 95-98% para letras que son simples (A, E, I, V), 90-95% para palabras básicas (Hola, Gracias), y 85-90% para frases que son complejas (Mi nombre es).

Tecnologías maduras: Para la implementación usamos frameworks open source que ya están bien establecidos (MediaPipe, TensorFlow, PyQt6) y estos nos dan estabilidad, soporte de la comunidad y actualizaciones que son continuas. Al usar software libre eliminamos los costos de licenciamiento y también permite que se puedan hacer adaptaciones en el futuro según las necesidades específicas que vayan surgiendo.

1.4.2. Impacto Proyectado

El impacto tanto social como económico que esperamos del sistema justifica de sobra la inversión que se haga en desarrollarlo e implementarlo.

Reducción de tiempos y costos: Calculamos que puede haber una reducción del 35% en el tiempo promedio que se tarda la atención para trámites donde están involucradas personas sordas. Esto significa ahorros operacionales que son significativos para las instituciones y también una mejora importante en la experiencia que tienen los usuarios.

Escalabilidad regional: El sistema se puede replicar en más de 150 instituciones tanto públicas como privadas que hay en Sinaloa, y tendría un impacto directo en aproximadamente 20,000 personas sordas a nivel estatal (esto aplicando la proporción nacional del 0.74% a los 2.7 millones de habitantes que tiene Sinaloa). Esta escalabilidad hace que el proyecto se convierta en una inversión de alto impacto social con costos marginales que van disminuyendo.

1.4.3. Innovación y Contribución Académica

Esta iniciativa aporta al desarrollo de soluciones tecnológicas que tienen enfoque social, y aplica conocimientos avanzados de inteligencia artificial a problemas reales de inclusión y accesibilidad. El hecho de que se enfoque específicamente en LSM representa una contribución que es original al campo, tomando en cuenta que la mayoría de las investigaciones se enfocan en lenguas de señas americana (ASL) o británica (BSL).

Contribución técnica: Desarrollar un pipeline completo que integre detección de keypoints, segmentación automática de señas y clasificación usando BiLSTM representa un avance técnico que es significativo en el procesamiento de LSM en tiempo real. La documentación detallada que hagamos del sistema va a permitir que otros investigadores lo puedan replicar y también mejorarlo.

Impacto académico: El proyecto va a generar conocimiento valioso sobre cuáles son las particularidades del LSM, metodologías para entrenar modelos cuando se tienen datasets limitados y técnicas de optimización para aplicaciones que funcionen en tiempo real. Estos resultados van a ser de utilidad para la comunidad académica tanto a nivel nacional como internacional.

Capítulo 2

2.Marco teórico

2.1. Introducción al Marco Teórico

El Marco Teórico constituye el soporte conceptual y científico sobre el cual se fundamenta este estudio. En este capítulo vamos a definir y analizar los principios, teorías y avances tecnológicos relevantes para diseñar un sistema de interpretación en tiempo real del Lenguaje de Señas Mexicano (LSM) mediante inteligencia artificial.

Empezaremos abordando los conceptos fundamentales relacionados con la discapacidad auditiva y los principios de comunicación inclusiva. Después exploraremos los aspectos formales del LSM, sus características lingüísticas y su variabilidad regional, algo que resulta bastante complejo cuando lo estudias a fondo. Posteriormente presentaremos los fundamentos teóricos de visión por computadora aplicada al reconocimiento de gestos, los principios del procesamiento de señales de audio para sistemas multimodales, y cuáles modelos de inteligencia artificial son más adecuados para procesar secuencias temporales.

También vamos a describir los fundamentos teóricos de arquitecturas de software para sistemas multimodales, principios de seguridad y autenticación en aplicaciones sensibles, y las bases conceptuales de sistemas de tiempo real. Finalmente analizaremos los criterios teóricos para seleccionar frameworks y tecnologías cuando desarrollas sistemas de reconocimiento gestual.

Revisar todos estos elementos nos permitirá establecer el contexto conceptual que necesitamos para entender la propuesta técnica y justificar por qué tomamos ciertas decisiones de diseño durante el desarrollo del sistema.

2.2. Discapacidad Auditiva y Comunicación Inclusiva

La discapacidad auditiva se puede definir como la disminución parcial o total de la capacidad para percibir sonidos, situación que afecta a una parte significativa de la población mundial. Según la Organización Mundial de la Salud (2021), más de 1,500 millones de personas viven con algún grado de pérdida auditiva, y esta cifra se proyecta que alcanzará 2,500 millones para 2050. Son números que impresionan cuando los ves así escritos.

Esta condición genera barreras de comunicación multidimensionales que van mucho más allá de simplemente no poder escuchar sonidos. Las personas sordas desarrollan una

identidad cultural propia, con lenguas visuales-gestuales que tienen estructuras gramaticales complejas y riqueza expresiva equivalente a cualquier lengua oral que conozcamos. Es algo realmente fascinante desde el punto de vista lingüístico.

2.2.1. Modelos Conceptuales de la Discapacidad

Modelo Médico vs Modelo Social: La forma en que entendemos la discapacidad auditiva ha ido evolucionando desde un modelo médico-rehabilitador hacia un modelo social que reconoce las barreras del entorno como el principal factor discapacitante (Oliver, 1996). Este cambio de perspectiva ha marcado una diferencia importante.

Tabla 2-1: Comparación de Modelos de Discapacidad

Aspecto	Modelo Médico	Modelo Social
Causa del problema	Deficiencia individual	Barreras sociales
Solución propuesta	Rehabilitación médica	Eliminación de barreras
Enfoque tecnológico	Dispositivos correctivos	Tecnologías inclusivas
Responsabilidad	Individual	Societal

2.2.2. Principios de Comunicación Inclusiva

La comunicación inclusiva se basa en el principio de **diseño universal**, que busca crear entornos, productos y servicios utilizables por todas las personas, en la mayor medida posible, sin necesidad de que tengan que usar adaptaciones o diseños especializados (Connell et al., 1997). Cuando lo piensas, es algo que debería ser obvio pero que en la práctica no siempre se aplica.

Principios del Diseño Universal aplicados a comunicación:

1. **Uso equitativo:** Útil para personas con capacidades diversas
2. **Flexibilidad de uso:** Acomoda preferencias y habilidades que tiene cada individuo
3. **Uso simple e intuitivo:** Fácil de entender, sin complicarse la vida
4. **Información perceptible:** Comunica información efectivamente
5. **Tolerancia al error:** Minimiza riesgos cuando alguien se equivoca
6. **Bajo esfuerzo físico:** Uso eficiente y confortable
7. **Espacio para acceso:** Tamaño apropiado para que la gente pueda aproximarse y usarlo

2.2.3. Métodos Tradicionales de Comunicación Inclusiva

Interpretación simultánea: Servicios profesionales que requieren personal certificado. Los estándares internacionales recomiendan 1 intérprete por cada 1,000 personas sordas, ratio que México francamente no alcanza - tenemos aproximadamente 800 intérpretes certificados para 2.3 millones de personas con discapacidad auditiva. La brecha es enorme.

Dispositivos de ayuda auditiva: Incluyen audífonos, implantes cocleares y sistemas de frecuencia modulada (FM). Qué tan efectivos sean varía según el tipo y grado de pérdida auditiva, y no todos los usuarios obtienen beneficios funcionales que realmente marquen la diferencia.

Sistemas de apoyo visual: Subtitulado, intérpretes en pantalla, y señalización visual que ayudan a acceder información auditiva. Estos han resultado ser útiles, aunque tienen sus propias limitaciones.

2.2.4. Limitaciones de los Enfoques Tradicionales

Las soluciones tradicionales tienen limitaciones sistémicas bastante claras:

- **Disponibilidad limitada:** Hay escasez de intérpretes profesionales
- **Costos elevados:** Los servicios especializados resultan económicamente inaccesibles para muchos
- **Dependencia temporal:** Necesitas programar con anticipación, lo cual no siempre es factible
- **Cobertura geográfica:** Se concentran en centros urbanos grandes, las áreas rurales quedan fuera

Estas limitaciones demuestran por qué necesitamos soluciones tecnológicas escalables que democratizen el acceso a herramientas de comunicación inclusiva. Es aquí donde la tecnología puede hacer una diferencia real.

2.3. Lenguaje de Señas Mexicano (LSM)

El Lenguaje de Señas Mexicano viene siendo la lengua natural que usa la comunidad sorda en México, reconocida oficialmente como lengua nacional mediante la Ley General para la Inclusión de las Personas con Discapacidad (2011). Posee gramática, sintaxis y estructuras léxicas propias que son bastante diferentes del español oral. Mucha gente no sabe que es una lengua completa por sí misma, con toda la complejidad que eso implica.

2.3.1. Estructura Lingüística del LSM

El LSM integra cinco parámetros formativos fundamentales para construir signos. Estos parámetros los estableció Stokoe (1960) y luego se adaptaron para las características específicas que tiene el LSM mexicano:

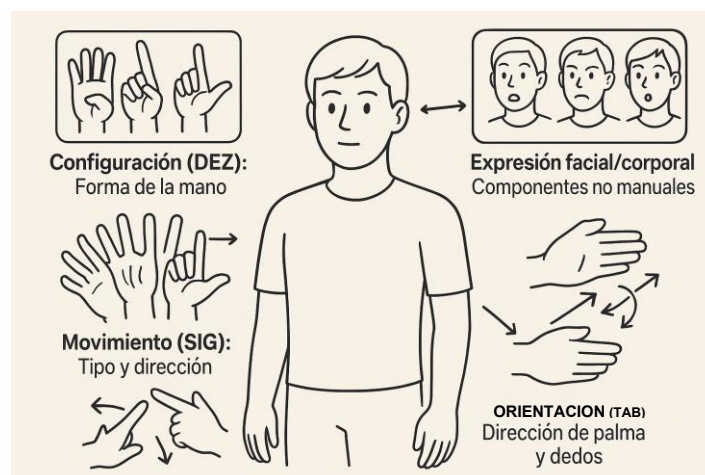


Figura 2-1: Parámetros Formativos del LSM

1. Configuración de la mano (dez): El LSM usa aproximadamente 45 configuraciones básicas - desde puños cerrados hasta extensiones específicas de dedos - y cada una tiene valor fonológico distintivo. Cuando empiezas a estudiar esto te das cuenta de que es mucho más complejo de lo que parece.

2. Ubicación (tab): Incluye zonas corporales específicas como la cabeza, torso, extremidades y también el espacio neutro frente al signante, con significado semántico asociado. La ubicación puede cambiar completamente el significado de una seña, lo cual es fascinante.

3. Movimiento (sig): Comprende el tipo, dirección, velocidad y repetición del desplazamiento. Puede ser rectilíneo, circular, vibratorio o combinaciones más complejas. Este parámetro resulta ser uno de los más difíciles de capturar con tecnología.

4. Orientación: La dirección hacia la cual apuntan la palma y los dedos. Es algo crucial para diferenciar signos que tienen configuración similar - un pequeño cambio de orientación y ya tienes una palabra completamente diferente.

5. Expresión facial y corporal: Componentes no manuales que aportan información gramatical, semántica y pragmática esencial. Sin estos elementos las señas pierden gran parte de su significado real, por eso son tan importantes.

2.3.2. Características Sintácticas Distintivas

Tabla 2-2: Comparación Sintáctica LSM vs Español

Característica	LSM	Español
Orden básico	Tópico-Comentario	Sujeto-Verbo-Objeto
Marcación temporal	Visual-espacial	Morfología verbal
Negación	Facial + manual	Partículas negativas
Interrogación	Facial + corporal	Entonación + partículas
Clasificadores	Sistema productivo	Ausente

Orden constituyente: El LSM presenta predominio del orden Tópico-Comentario sobre la estructura Sujeto-Verbo-Objeto que usa el español, reflejando principios de organización visual-espacial del discurso. Es una diferencia fundamental que afecta cómo se estructura toda la comunicación.

Clasificadores: Sistema de movimientos de mano que representan forma, tamaño, textura o movimiento de objetos a los que se refiere, constituyendo un mecanismo gramatical bastante productivo. Cuando ves esto en acción te das cuenta de lo sofisticado que puede ser el LSM.

Rasgos no manuales: Las expresiones faciales, movimientos de cabeza y postura corporal cumplen funciones gramaticales específicas que son equivalentes a la entonación en lenguas orales. Sin estos elementos, las señas quedan como "planas" y pierden mucho significado.

2.3.3. Variabilidad Diatópica y Diastrática

Una característica distintiva del LSM es su variabilidad regional y sociocultural. Investigaciones del Instituto Nacional para la Educación de los Adultos (INEA, 2019) documentan diferencias significativas relacionadas con:

Factores de variación:

- **Edad:** Diferencias generacionales en adopción de neologismos
- **Nivel educativo:** Influencia de educación formal en estandarización
- **Contexto social:** Variaciones entre LSM doméstico, educativo y comunitario
- **Contacto lingüístico:** Influencia del español en zonas de mayor integración

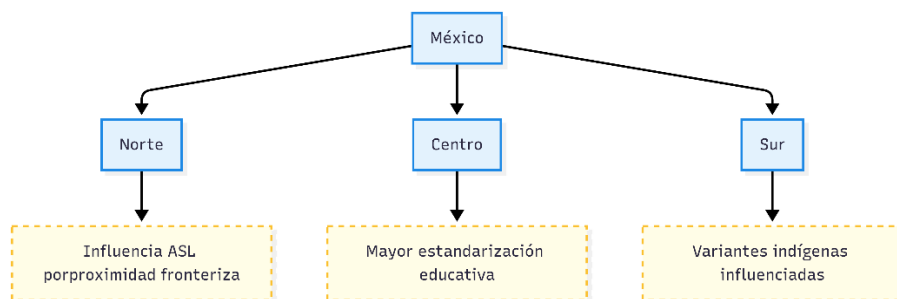


Figura 2-2: Mapa Conceptual de Variación en LSM

2.3.4. LSM en Contextos Institucionales

El vocabulario especializado para trámites institucionales presenta características específicas:

- **Neologismos técnicos:** Adaptación de terminología administrativa
- **Regionalismo institucional:** Variaciones según tipos de instituciones
- **Código switching:** Alternancia LSM-español deletreado
- **Registro formal:** Modificaciones estilísticas para contextos oficiales

2.3.5. Fundamentos de Visión por Computadora para Reconocimiento Gestual

La visión por computadora constituye una rama de la inteligencia artificial que permite a las máquinas interpretar y procesar información visual extraída de imágenes o secuencias de video. En el contexto del reconocimiento de lenguaje de señas, su función principal consiste en detectar, extraer y procesar características relevantes de los movimientos gestuales.

2.3.6. Pipeline Conceptual de Procesamiento Visual

El procesamiento de gestos mediante visión por computadora sigue un pipeline estructurado que transforma datos visuales brutos en representaciones numéricas procesables:

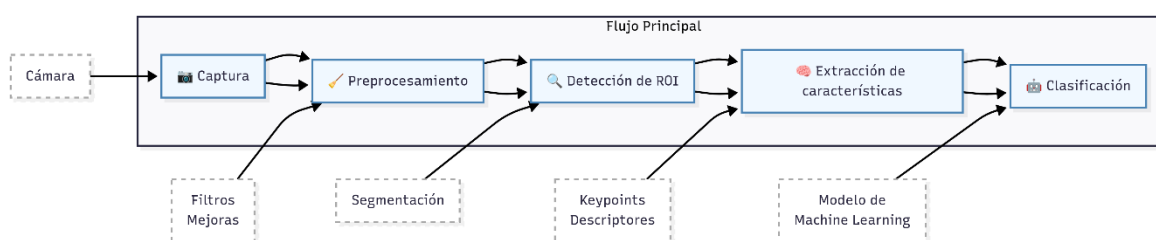


Figura 2-3: Pipeline General de Procesamiento Visual

Etapas del pipeline:

1. **Adquisición:** Captura de secuencias de video usando sensores RGB. Es el paso más básico pero fundamental.
2. **Preprocesamiento:** Mejora de calidad, normalización y reducción de ruido. Aquí es donde limpiamos todo lo que puede interferir con el análisis.
3. **Segmentación:** Identificación de regiones de interés como manos, cara, cuerpo. La computadora aprende a enfocarse en lo que realmente importa.
4. **Extracción:** Conversión a descriptores numéricos procesables. Básicamente traducir lo visual a números que la máquina pueda manejar.
5. **Clasificación:** Interpretación semántica mediante algoritmos de ML. Aquí es donde finalmente "entiende" qué seña se está haciendo.

2.3.7. Técnicas de Representación de Características

Keypoints (Puntos Clave): Representación de articulaciones y puntos de referencia anatómicos mediante coordenadas espaciales precisas. Esta aproximación permite describir la configuración corporal con vectores de baja dimensionalidad, pero que tienen alta información semántica. Es como mapear el esqueleto humano en puntos específicos.

Descriptores de Movimiento: Características dinámicas derivadas de secuencias temporales. Aquí es donde la cosa se pone interesante:

- **Velocidad angular:** Cambios en orientación de segmentos corporales. Qué tan rápido se mueven las diferentes partes.

- **Aceleración:** Variaciones en la velocidad de movimiento. No es lo mismo un movimiento constante que uno que acelera o desacelera.
- **Trayectorias:** Rutas espaciales que describen puntos específicos. Como el camino que sigue la mano durante una seña.
- **Diferencias temporales:** Cambios entre frames consecutivos. La computadora compara frame por frame para detectar movimiento.

Tabla 2-3: Comparativa de Técnicas de Extracción de Características

Técnica	Dimensionalidad	Robustez	Costo Computacional	Aplicabilidad LSM
Keypoints	Baja (42-126)	Alta	Bajo	Excelente
Optical Flow	Media (N×2)	Media	Medio	Buena
Descriptores HOG	Alta (3780+)	Media	Alto	Regular
CNN Features	Muy Alta (512+)	Alta	Muy Alto	Buena

2.3.8. Desafíos Técnicos en Reconocimiento Gestual

Variabilidad en Condiciones Ambientales:

- **Iluminación:** Los cambios en intensidad y dirección de la luz. Este es un problemón real - imagínate que estás en una oficina con ventanas grandes y nubes que van y vienen, o luces fluorescentes que parpadean. La detección se vuelve un caos total.
- **Fondos complejos:** Cuando hay interferencia con elementos visuales del entorno. Si atrás tienes gente caminando, pósters en la pared, o cualquier cosa que se mueva, la computadora se marea tratando de distinguir qué son las manos y qué es ruido de fondo.
- **Resolución:** Las limitaciones que tiene el hardware de captura. Una cámara de 200 pesos no va a captar el mismo detalle que una profesional, y eso se nota mucho en la precisión del reconocimiento.

Oclusiones y Auto-occlusiones:

- **Oclusión parcial:** Cuando hay superposición temporal de manos o algo externo interfiere. Por ejemplo, si alguien camina entre la cámara y el usuario justo en el momento clave.
- **Auto-occlusión:** Situaciones donde una mano tapa parcialmente la otra. Esto pasa constantemente en señas complejas y es una de las causas principales de errores del sistema.
- **Oclusión corporal:** Interferencia con otros elementos anatómicos. A veces el propio cuerpo del usuario bloquea la vista de las manos desde cierto ángulo.

Variabilidad Inter e Intra-personal:

- **Diferencias individuales:** Variaciones en tamaño, velocidad y estilo gestual entre diferentes personas. No todos tenemos el mismo tamaño de manos ni hacemos los movimientos a la misma velocidad.

- **Consistencia temporal:** Variaciones en cómo ejecuta las señas el mismo usuario en diferentes momentos. Ni la misma persona hace las señas exactamente igual cada vez que las repite.
- **Factores contextuales:** Cómo influye el estado emocional y la fatiga. Si estás nervioso, cansado o apurado, las señas salen diferentes aunque no te des cuenta.

2.3.9. Métricas de Evaluación en Visión por Computadora

Precisión de Detección: El porcentaje de keypoints que detecta correctamente dentro de un umbral de error aceptable (normalmente entre 5-10 píxeles). Es como medir qué tan buena puntería tiene el sistema.

Robustez Temporal: La capacidad que tiene para mantener tracking consistente durante secuencias temporales completas, medida a través de qué tan continuas son las trayectorias. En cristiano: que no pierda el seguimiento a mitad de una seña.

Eficiencia Computacional: Se mide en frames por segundo (FPS) que puede sostener en hardware específico. Esto es crucial para aplicaciones de tiempo real porque si va muy lento, la comunicación se vuelve frustrante.

2.4. Procesamiento de Señales de Audio para Sistemas Multimodales

El procesamiento de audio en sistemas de comunicación inclusiva necesita técnicas especializadas para capturar, segmentar y analizar señales de voz. Esto resulta especialmente importante en aplicaciones bidireccionales donde usuarios sordos y oyentes tienen que interactuar entre sí.

2.4.1. Fundamentos de Digitalización de Audio

Teorema de Muestreo de Nyquist: La frecuencia de muestreo tiene que ser como mínimo el doble de la frecuencia máxima de la señal para evitar aliasing. Para voz humana (que va de 0 a 8kHz), necesitas por lo menos 16kHz de muestreo. Esto suena súper técnico, pero básicamente es la fórmula matemática para convertir sonido analógico a digital sin que se pierda información por el camino.

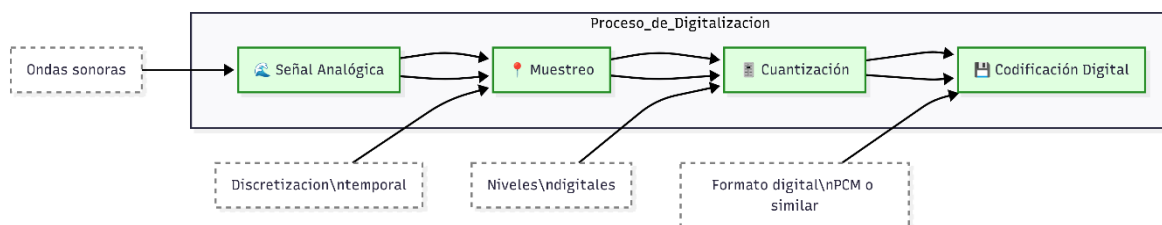


Figura 2-4: Proceso de Digitalización de Audio

Parámetros críticos:

- **Frecuencia de muestreo:** Va de 16kHz a 48kHz, dependiendo de qué tanto necesites. Para conversación básica con 16kHz te alcanza perfectamente, pero si andas buscando calidad profesional entonces sí necesitas llegar a los 48kHz. Es como la diferencia entre una llamada telefónica y un estudio de grabación.
- **Profundidad de bits:** Entre 16 y 24 bits para tener un rango dinámico que sirva. Si usas menos bits vas a terminar con audio que suena como radio AM de los años 80 - horrible.
- **Canales:** Mono contra estéreo, dependiendo de si necesitas ubicar de dónde viene el sonido. Para reconocimiento de voz normalmente con mono es más que suficiente, no vale la pena complicarse.

2.4.2. Técnicas de Segmentación de Voz

Detección de Actividad Vocal (VAD): Son algoritmos que distinguen cuándo hay voz y cuándo hay silencio. Suena obvio pero es fundamental para sistemas de reconocimiento automático - imagínate si la computadora trata de interpretar cada ruidito como si fuera habla.

Métodos de VAD:

1. **Basados en energía:** Análisis de qué tan fuerte está la señal. Es el método más básico pero efectivo - si hay sonido fuerte, probablemente alguien esté hablando. Simple pero funciona.
2. **Basados en frecuencia:** Aquí analizan qué frecuencias están presentes porque la voz humana tiene características específicas que la distinguen de otros ruidos. Es más sofisticado que solo medir volumen.
3. **Basados en ML:** Clasificadores que entrenaron con un montón de características acústicas. Estos son más inteligentes porque aprenden patrones complejos de cómo suena la voz humana vs ruido de tráfico, música de fondo, etc.
4. **Híbridos:** Combinan varios métodos porque ninguno es perfecto para todas las situaciones. A veces necesitas la simplicidad del método de energía combinado con la sofisticación del ML para cubrir todos los casos raros que se presentan.

Tabla 2-4: Comparativa de Métodos VAD

Método	Precisión	Latencia	Robustez a Ruido	Complejidad
Energía	85-90%	<10ms	Baja	Muy Baja
Espectral	90-95%	20-50ms	Media	Baja
ML	95-98%	50-100ms	Alta	Alta
Híbrido	96-99%	30-80ms	Muy Alta	Media

2.4.3. Preprocesamiento de Señales de Audio

Filtrado de Ruido: Técnicas para eliminar interferencias ambientales que degradan la calidad del reconocimiento. En la práctica este paso es crucial porque siempre hay ruido de fondo interfiriendo en entornos reales.

- **Filtros paso-banda:** Eliminan frecuencias fuera del rango vocal. Lo que hacen es restringir el análisis únicamente a frecuencias donde se produce voz humana.
- **Supresión espectral:** Reducción de ruido estacionario. Hablamos del zumbido constante de equipos de aire acondicionado o tráfico vehicular que, aunque no lo notes, confunde bastante al sistema.
- **Filtros adaptativos:** Se ajustan dinámicamente según las características del entorno. Son filtros que van aprendiendo y modificando su comportamiento sobre la marcha.

Normalización: Ajuste de niveles de amplitud para mantener consistencia entre diferentes fuentes de audio. Sin este proceso tendrías grabaciones con volúmenes completamente dispares.

- **Normalización de ganancia:** Ajusta el nivel promedio de la señal. Es similar a ecualizar el volumen de diferentes pistas de audio.
- **Compresión dinámica:** Reduce el rango dinámico para que los sonidos suaves se escuchen mejor y los fuertes no saturen el sistema.
- **Control automático de ganancia (AGC):** La computadora ajusta el volumen automáticamente durante la captura. Es especialmente útil cuando los usuarios hablan a diferentes distancias del micrófono.

2.4.4. Calidad de Audio para Reconocimiento Automático

Métricas de Calidad:

- **Signal-to-Noise Ratio (SNR):** Relación entre señal útil y ruido de fondo. Entre más alto sea este valor, mejor se distingue la voz del ruido ambiente.
- **Total Harmonic Distortion (THD):** Medida de distorsión armónica presente en la señal. Te dice qué tanto se degrada el sonido durante todo el proceso de captura y digitalización.
- **Word Error Rate (WER):** Métrica específica para sistemas de reconocimiento. Cuenta cuántas palabras malinterpreta del total que pronunciaste.

Requerimientos mínimos para aplicaciones LSM:

SNR superior a 20dB para reconocimiento confiable, ya que valores menores comprometen significativamente la precisión. THD menor al 1% para preservar características espectrales importantes. Latencia inferior a 100ms para que la experiencia se perciba como natural en lugar de artificial.

2.5. Síntesis de Voz y Text-to-Speech

La síntesis de voz constituye el proceso de generar habla humana artificial a partir de texto. Es un componente esencial en sistemas de comunicación bidireccional para facilitar la interacción entre usuarios sordos y oyentes. En términos simples: hacer que la máquina hable.

2.5.1. Fundamentos de Síntesis de Voz

Componentes de un Sistema TTS:

1. **Análisis de texto:** Procesamiento del texto de entrada donde el sistema interpreta qué debe pronunciar.
2. **Análisis lingüístico:** Conversión a representación fonética. Aquí determina cómo pronunciar correctamente cada palabra, lo cual presenta mayor complejidad de la que inicialmente parece.
3. **Síntesis acústica:** Generación de la señal de audio real. Esta etapa produce efectivamente el sonido audible.
4. **Post-procesamiento:** Mejora y filtrado de la salida para que resulte más natural y menos robótica.

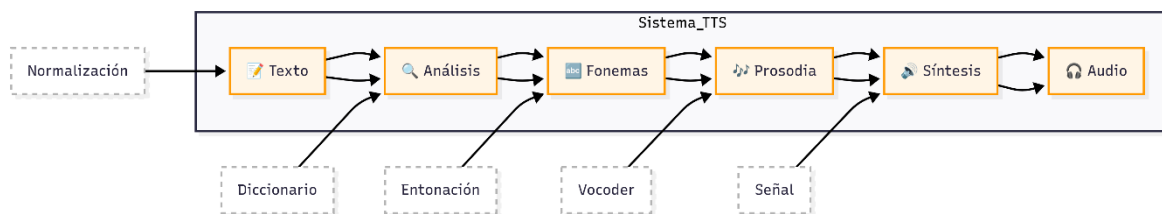


Figura 2-5: Arquitectura Conceptual de Sistema TTS

2.5.2. Tipos de Síntesis de Voz

Síntesis Concatenativa: Utiliza grabaciones de voz humana segmentadas y concatenadas. En la práctica toma pedacitos de grabaciones reales y los junta para formar palabras nuevas:

- **Ventajas:** Alta naturalidad, inteligibilidad excelente
- **Desventajas:** Requiere bases de datos gigantescas de voz grabada
- **Aplicaciones:** Sistemas comerciales, aplicaciones críticas donde la calidad es lo más importante

Síntesis Paramétrica: Genera voz mediante modelos matemáticos de producción vocal. Lo que hace es simular matemáticamente cómo funciona la voz humana:

- **Ventajas:** Menor requerimiento de almacenamiento, flexibilidad en la implementación
- **Desventajas:** El sonido resulta más artificial comparado con otros métodos
- **Aplicaciones:** Sistemas embebidos, aplicaciones donde tienes recursos limitados

Síntesis Neural: Utiliza redes neuronales profundas para modelar la producción de voz. Es la tecnología más reciente en este campo:

- **Ventajas:** Balance entre naturalidad y flexibilidad, que es lo que buscas idealmente
- **Desventajas:** Los requerimientos computacionales son bastante elevados
- **Aplicaciones:** Sistemas modernos de alta calidad donde puedes permitirte el costo computacional

Tabla 2-5: Comparativa de Tecnologías TTS

Tipo	Naturalidad	Inteligibilidad	Recursos	Latencia	Flexibilidad
Concatenativa	Muy Alta	Excelente	Altos	Baja	Baja
Paramétrica	Media	Buena	Bajos	Muy Baja	Alta
Neural	Alta	Muy Buena	Muy Altos	Media	Alta

2.5.3. Métricas de Calidad en TTS

Mean Opinion Score (MOS): Es una evaluación subjetiva de calidad que usa una escala del 1 al 5:

- **5:** Excelente - no puedes distinguirlo de voz natural
- **4:** Bueno - alta calidad aunque con algunos artefactos menores
- **3:** Aceptable - se entiende pero claramente es sintético
- **2:** Pobre - se entiende pero con esfuerzo
- **1:** Malo - definitivamente ininteligible

Métricas Objetivas:

- **Word Error Rate (WER):** Cuántos errores comete en reconocimiento de voz sintética
- **Mel-Cepstral Distortion (MCD):** Qué tan lejos está espectralmente de voz natural
- **F0 RMSE:** El error que tiene en frecuencia fundamental, o sea el pitch

2.5.4. Consideraciones para Aplicaciones de Accesibilidad

Parámetros Prosódicos Optimizados:

- **Velocidad:** Entre 110 y 150 palabras por minuto funciona mejor para comprensión máxima
- **Articulación:** Pronunciación clara de consonantes y vocales, obvio
- **Pausas:** Que marque bien las fronteras sintácticas

Adaptación Cultural:

- **Acento regional:** Seleccionar variantes del español mexicano apropiadas
- **Vocabulario especializado:** Pronunciar correctamente términos técnicos, que no es fácil
- **Entonación:** Patrones prosódicos que sean apropiados para contexto formal

2.6. Inteligencia Artificial para Reconocimiento de Secuencias Temporales

Aplicar inteligencia artificial al reconocimiento de gestos de lenguaje de señas requiere arquitecturas especializadas. Estas arquitecturas deben poder modelar dependencias temporales complejas en secuencias de datos multidimensionales.

2.6.1. Evolución de Arquitecturas para Secuencias

Cómo ha evolucionado históricamente:

1. **Redes Feed-Forward (1950s-1980s):** Estaban limitadas a patrones estáticos
2. **Redes Recurrentes (1980s-1990s):** Aquí introdujeron memoria temporal
3. **LSTM/GRU (1990s-2010s):** La solución al problema del gradiente desvaneciente
4. **Attention/Transformers (2010s-presente):** Paralelización y contexto global

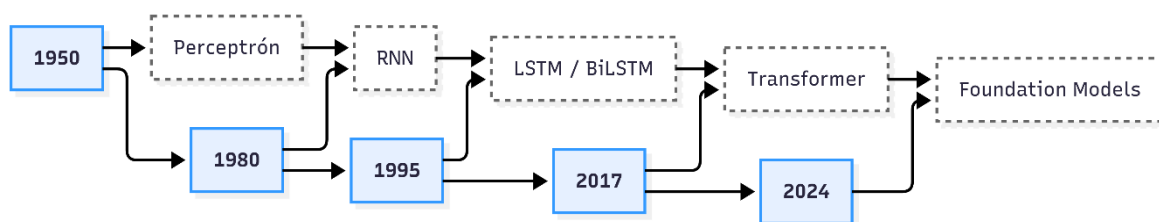


Figura 2-6: Evolución de Arquitecturas para Secuencias Temporales

2.6.2. Redes Neuronales Recurrentes (RNN)

Fundamentos conceptuales: Las RNN introducen conexiones recurrentes que permiten mantener información de estados previos. Esto habilita el procesamiento de secuencias de longitud variable, algo que las redes anteriores no podían hacer.

Ecuaciones fundamentales:

$$h_t = f(W_h h * h_{t-1} + W_x h * x_t + b_h) y_t = W_h y * h_t + b_y$$

Problema del desvanecimiento del gradiente: En secuencias largas, los gradientes se van desvaneciendo exponencialmente. Esto limita bastante la capacidad de aprender dependencias de largo alcance, que es justamente lo que necesitas para reconocer señas.

1.1.1. Long Short-Term Memory (LSTM)

Arquitectura de compuertas: Las redes LSTM resuelven el problema del desvanecimiento mediante un mecanismo de compuertas que regula cómo fluye la información:

Compuertas fundamentales:

1. **Forget Gate:** Determina qué información descartar del estado celular. Básicamente decide qué olvidar.
2. **Input Gate:** Controla qué nuevos valores se van a almacenar en el estado.
3. **Output Gate:** Regula qué partes del estado se usan para la salida final.

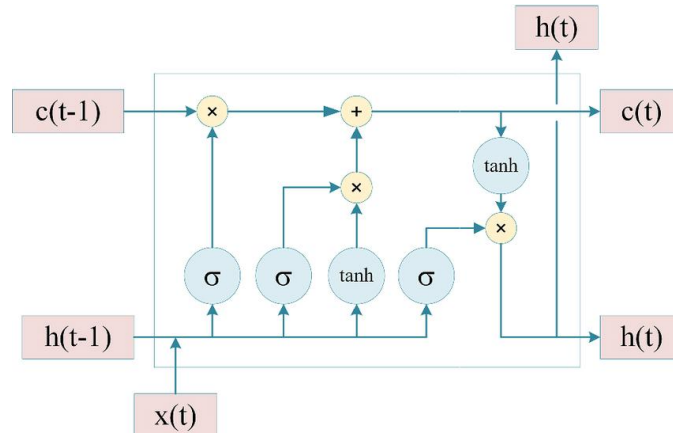


Figura 2-7: Arquitectura Conceptual de Neurona LSTM

Ventajas para reconocimiento gestual:

- Capacidad de modelar dependencias de largo alcance
- Robustez ante variaciones en velocidad de ejecución
- Manejo efectivo de secuencias de longitud variable

2.6.3. Bidirectional LSTM (BiLSTM)

Fundamentos teóricos: Las redes BiLSTM procesan secuencias en ambas direcciones temporales, capturando dependencias tanto de contexto pasado como futuro.

Arquitectura bidireccional:

$$\vec{h}_t = \text{LSTM}_{\text{forward}}(x_1, x_2, \dots, x_t)$$

$$\overleftarrow{h}_t = \text{LSTM}_{\text{backward}}(x_T, x_{T-1}, \dots, x_t)$$

$$h_t = \text{concatenate}(\vec{h}_t, \overleftarrow{h}_t)$$

Tabla 2-6: Comparativa de Arquitecturas Recurrentes

Arquitectura	Contexto	Complejidad	Precisión Típica	Aplicabilidad LSM
RNN Vanilla	Unidireccional	Baja	70-80%	Limitada
LSTM	Unidireccional	Media	85-90%	Buena
BiLSTM	Bidireccional	Media-Alta	90-95%	Excelente
GRU	Unidireccional	Media	85-90%	Buena

Ventajas específicas para LSM:

- Acceso a información contextual completa de la secuencia gestual
- Mejor desambiguación de gestos con fases iniciales similares
- Robustez ante variaciones temporales en ejecución

2.6.4. Técnicas de Regularización y Optimización

Dropout: Técnica que aleatoriamente desactiva neuronas durante entrenamiento para prevenir overfitting:

- **Dropout estándar:** Aplicado a capas densas
- **Recurrent Dropout:** Específico para conexiones recurrentes
- **Variational Dropout:** Consistencia en máscaras a través del tiempo

Batch Normalization: Normalización de activaciones para estabilizar entrenamiento:

- Acelera convergencia
- Permite tasas de aprendizaje más altas
- Actúa como regularizador implícito

Gradient Clipping: Prevención de explosión de gradientes mediante limitación de normas:

$$[\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V] [\text{Si } \|\nabla_\theta\| > \tau, \quad \nabla_\theta \leftarrow \nabla_\theta \cdot \frac{\tau}{\|\nabla_\theta\|}]$$

2.6.5. Métricas de Evaluación para Clasificación Temporal

Métricas de clasificación:

- **Accuracy:** Porcentaje de predicciones correctas
- **Precision/Recall:** Métricas por clase para datasets desbalanceados
- **F1-Score:** Media armónica de precision y recall
- **Confusion Matrix:** Análisis detallado de errores por clase

Métricas temporales:

- **Temporal Accuracy:** Precisión considerando ventanas temporales
- **Segment-level Metrics:** Evaluación a nivel de segmentos gestuales
- **Frame-level Metrics:** Evaluación frame por frame

2.7. Arquitecturas de Software para Sistemas Multimodales

Diseñar sistemas que integran múltiples modalidades de entrada (video, audio) y salida (texto, voz) requiere arquitecturas especializadas. Estas arquitecturas tienen que garantizar sincronización, escalabilidad y mantenibilidad - algo que definitivamente no es sencillo cuando tienes diferentes tipos de datos fluyendo al mismo tiempo por todos lados.

2.7.1. Patrones de Diseño para Aplicaciones de Tiempo Real

Patrón Pipeline: Procesamiento secuencial de datos a través de etapas especializadas. Funciona como una línea de ensamblaje:

- **Ventajas:** Simplicidad conceptual, facilidad para hacer debugging
- **Desventajas:** La latencia se va acumulando conforme pasa por cada etapa, y si una etapa anda lenta pues se convierte en cuello de botella para todo
- **Aplicabilidad:** Funciona bien en sistemas donde los datos van fluyendo en una sola dirección

Patrón Producer-Consumer: Separación entre quien genera los datos y quien los consume. Básicamente uno produce y otro procesa:

- **Ventajas:** Desacoplamiento temporal, puede manejar picos de carga sin problemas
- **Desventajas:** La sincronización se vuelve compleja, gestión de buffers que puede dar dolores de cabeza
- **Aplicabilidad:** Sistemas donde las tasas de producción y consumo varían bastante

Patrón Event-Driven: Arquitectura que se basa en eventos asincrónicos:

- **Ventajas:** Escalabilidad, modularidad, buena responsividad
- **Desventajas:** El debugging se complica mucho, pueden aparecer condiciones de carrera que son un nightmare para detectar
- **Aplicabilidad:** Sistemas donde tienes múltiples fuentes de eventos

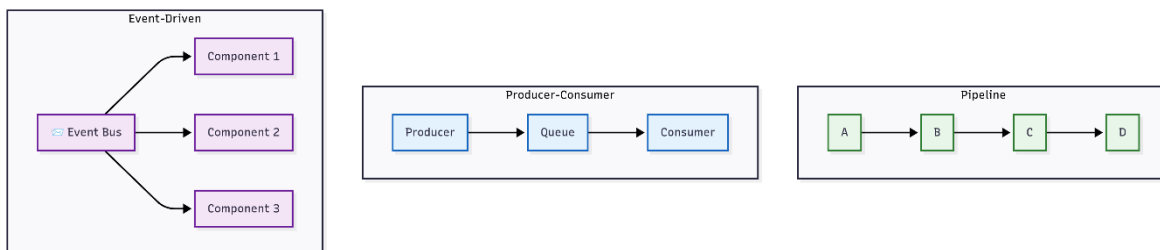


Figura 2-8: Patrones Arquitecturales para Sistemas Multimodales

2.7.2. Arquitecturas Cliente-Servidor vs Standalone

Tabla 2-7: Comparativa Arquitectural para Sistemas LSM

Aspecto	Cliente-Servidor	Standalone
Latencia	50-500ms (red)	10-100ms (local)
Privacidad	Dependiente de implementación	Completa
Escalabilidad	Alta (horizontal)	Limitada (vertical)
Mantenimiento	Centralizado	Distribuido
Disponibilidad	Dependiente de conectividad	Offline completo
Costo computacional	Compartido	Individual

Consideraciones para aplicaciones LSM:

- **Sensibilidad de datos:** Datos biométricos requieren máxima privacidad
- **Latencia crítica:** Comunicación natural requiere <200ms
- **Disponibilidad:** Acceso garantizado independiente de conectividad
- **Escalabilidad:** Balance entre rendimiento y recursos

2.7.3. Frameworks de Desarrollo para Aplicaciones Desktop

Criterios de selección para sistemas multimodales:

1. **Acceso a hardware:** Cámara, micrófono, altavoces
2. **Capacidades multimedia:** Procesamiento de video/audio en tiempo real
3. **Rendimiento:** Latencias mínimas para interacción natural
4. **Multiplataforma:** Compatibilidad con sistemas operativos principales
5. **Ecosistema:** Disponibilidad de bibliotecas especializadas

Tabla 2-8: Comparativa de Frameworks Desktop

Framework	Lenguaje	Rendimiento	Multimedia	Curva Aprendizaje	Ecosistema
Qt	C++/Python	Excelente	Excelente	Media-Alta	Muy Amplio
Electron	JavaScript	Bueno	Bueno	Baja	Amplio
.NET WPF	C#	Muy Bueno	Bueno	Media	Amplio
Tkinter	Python	Regular	Limitado	Baja	Limitado
JavaFX	Java	Bueno	Bueno	Media	Amplio

2.7.4. Gestión de Concurrencia en Aplicaciones Multimedia

Threading Models:

- **Single-threaded:** Procesamiento secuencial, riesgo de bloqueo de UI
- **Multi-threaded:** Paralelización de tareas, complejidad de sincronización
- **Actor Model:** Comunicación por mensajes, eliminación de estado compartido
- **Event Loop:** Procesamiento asíncrono de eventos, eficiencia en I/O

Consideraciones específicas para LSM:

- **Captura de video:** Thread dedicado para mantener framerate
- **Procesamiento ML:** Thread separado para evitar bloqueo de UI

- **Audio:** Thread de alta prioridad para latencia mínima
- **Interfaz:** Thread principal para responsividad de usuario

2.8. Sistemas de Tiempo Real

Los sistemas de tiempo real para reconocimiento de LSM deben cumplir restricciones temporales estrictas para mantener la naturalidad de la comunicación humana.

2.8.1. Clasificación de Sistemas de Tiempo Real

Hard Real-Time: Restricciones temporales estrictas cuya violación causa falla del sistema:

- **Aplicaciones:** Sistemas críticos de seguridad
- **Tolerancia:** 0% de violaciones aceptables
- **Aplicabilidad LSM:** No requerido (comunicación no crítica)

Soft Real-Time: Restricciones temporales importantes pero tolerantes a violaciones ocasionales:

- **Aplicaciones:** Multimedia, comunicación interactiva
- **Tolerancia:** <5% de violaciones aceptables
- **Aplicabilidad LSM:** Ideal para comunicación natural

Tabla 2-10: Requisitos Temporales para Comunicación Humana

Modalidad	Latencia Imperceptible	Latencia Aceptable	Latencia Problemática
Audio	<20ms	20-150ms	>150ms
Video	<16ms (60fps)	16-33ms (30fps)	>33ms
Gestos LSM	<50ms	50-200ms	>200ms
Texto	<100ms	100-500ms	>500ms

2.8.2. Técnicas de Optimización de Rendimiento

Optimización algorítmica:

- **Complejidad temporal:** Usar algoritmos sub-lineales cuando se pueda hacer
- **Early stopping:** Parar antes cuando ya sabes que las predicciones son confiables
- **Approximation algorithms:** El clásico trade-off entre qué tan preciso vs qué tan rápido
- **Parallel processing:** Repartir la carga entre varios procesos para que no se atore

Optimización de hardware:

- **CPU optimization:** Vectorización, algoritmos que no peleen con el cache
- **GPU acceleration:** Paralelización masiva, especialmente útil para ML
- **Memory optimization:** Minimizar asignaciones dinámicas que fragmentan todo
- **I/O optimization:** Buffering y streaming eficientes para no estar perdiendo tiempo

2.8.3. Gestión de Recursos en Sistemas Multimedia

Scheduling de tareas:

- **Priority-based:** Las tareas críticas obviamente van primero
- **Rate monotonic:** Prioridad inversa al período de ejecución
- **Earliest deadline first:** La prioridad depende de qué tan cerca ande el deadline
- **Round-robin:** Distribución equitativa usando quantum de tiempo fijo

Resource allocation:

- **CPU affinity:** Asignar threads a cores específicos para mejor rendimiento
- **Memory pools:** Pre-asignar memoria para evitar que se fragmente después
- **Network QoS:** Priorizar tráfico crítico sobre todo lo demás
- **Power management:** Balancear entre rendimiento y cuánta batería gastas

2.9. Frameworks y Bibliotecas para Desarrollo de Sistemas LSM

Seleccionar las herramientas tecnológicas es una decisión crítica. Impacta directamente el rendimiento, mantenibilidad y escalabilidad del sistema que vayas a desarrollar. No es algo que puedas decidir sin pensarle bien.

2.9.1. Criterios de Selección de Tecnologías

Factores técnicos:

1. **Rendimiento:** Latencia, throughput, qué tan eficiente anda con los recursos
2. **Funcionalidad:** Qué tanto cubre los requerimientos específicos que tienes en mente
3. **Estabilidad:** Madurez del framework, confiabilidad, qué tan bien arreglan bugs
4. **Documentación:** Calidad y qué tan completa está toda la documentación
5. **Comunidad:** Tamaño de la comunidad, qué tan activa anda, soporte que dan

Factores estratégicos:

1. **Licenciamiento:** Que sea compatible con el modelo de negocio que manejas
2. **Vendor lock-in:** Qué tanta independencia mantienes de proveedores específicos
3. **Roadmap:** Que vaya alineado con la evolución futura de tu proyecto
4. **Ecosistema:** Que se integre decentemente con otras herramientas que ya usas
5. **Skill availability:** Que haya desarrolladores por ahí que sepan usarlo bien

2.9.2. Comparativa de Bibliotecas de Visión por Computadora

Tabla 2-11: Frameworks de Detección de Pose/Keypoints

Framework	Precisión	Rendimiento	Facilidad Uso	Licencia	Soporte
MediaPipe	95%+	Excelente	Alta	Apache 2.0	Google

OpenPose	93%+	Bueno	Media	Custom	CMU
PoseNet	90%+	Muy Bueno	Alta	Apache 2.0	TensorFlow
AlphaPose	95%+	Bueno	Baja	GPL 3.0	Academia
BlazePose	94%+	Excelente	Media	Apache 2.0	Google

2.9.3. Frameworks de Deep Learning

Consideraciones para selección:

- **API design:** Facilidad de desarrollo y debugging
- **Performance:** Velocidad de entrenamiento e inferencia
- **Deployment:** Opciones de producción y optimización
- **Ecosystem:** Disponibilidad de modelos pre-entrenados
- **Hardware support:** Compatibilidad con aceleradores

Tabla 2-12: Comparativa de Frameworks de ML

Framework	Facilidad Uso	Rendimiento	Despliegue	Ecosistema	Aplicabilidad LSM
TensorFlow	Media	Excelente	Excelente	Muy Amplio	Excelente
PyTorch	Alta	Excelente	Bueno	Amplio	Muy Buena
Keras	Muy Alta	Bueno	Bueno	Amplio	Excelente
ONNX	Baja	Excelente	Excelente	Medio	Buena
JAX	Media	Excelente	Medio	Pequeño	Buena

2.9.4. Frameworks de Desarrollo de Interfaces

Criterios específicos para aplicaciones LSM:

- **Multimedia capabilities:** Soporte nativo para video/audio
- **Real-time performance:** Latencias mínimas en UI
- **Accessibility:** Compatibilidad con tecnologías de asistencia
- **Cross-platform:** Funcionamiento en múltiples OS
- **Hardware access:** Acceso directo a dispositivos

Recomendaciones por tipo de aplicación:

- **Prototipos rápidos:** Tkinter, Streamlit
- **Aplicaciones comerciales:** Qt, Electron, .NET
- **Aplicaciones web:** React, Vue.js, Angular
- **Aplicaciones móviles:** Flutter, React Native, Xamarin

2.10. Conclusiones del Marco Teórico

En este capítulo hemos establecido todo el fundamento conceptual que necesitamos para desarrollar un sistema de interpretación en tiempo real del Lenguaje de Señas Mexicano usando inteligencia artificial. Cubrimos desde principios básicos de comunicación inclusiva hasta las tecnologías más avanzadas para reconocer secuencias temporales.

Cuando analizamos la discapacidad auditiva desde una perspectiva social, quedó claro que hace falta urgentemente herramientas tecnológicas que no nada más complementen lo que ya existe, sino que democratizen el acceso a métodos de comunicación inclusiva. Al explorar las características lingüísticas del LSM descubrimos su complejidad estructural y riqueza expresiva, lo cual nos da la base conceptual para desarrollar algo que respete la diversidad cultural de la comunidad sorda mexicana.

Todo lo que revisamos sobre fundamentos de visión por computadora y procesamiento de señales nos dio el marco técnico para entender tanto los desafíos como las oportunidades en reconocimiento automático de gestos. Cuando revisamos las arquitecturas de inteligencia artificial, esto justificó por qué tiene sentido usar aproximaciones basadas en redes recurrentes bidireccionales para modelar secuencias gestuales que cambian en el tiempo.

Las consideraciones sobre arquitecturas de software, seguridad y sistemas de tiempo real nos dieron los principios de diseño que necesitamos para crear sistemas que sean robustos, seguros y eficientes. El análisis que hicimos de frameworks y tecnologías nos proporcionó el contexto para tomar decisiones informadas cuando llegue el momento de implementar.

Todo este marco teórico sólido va a ser la base conceptual sobre la cual construiremos el desarrollo práctico del sistema, garantizando que las decisiones técnicas que tomemos estén respaldadas por principios científicos establecidos y las mejores prácticas que maneja la industria.

Capítulo 3

3. Estado del arte

3.1. Introducción al estado del arte

En este capítulo se revisa la literatura científica y las soluciones tecnológicas más relevantes relacionadas con el reconocimiento automático del Lenguaje de Señas Mexicano (LSM), la segmentación de gestos en video, los modelos de inteligencia artificial aplicados a secuencias visuales y de audio, así como las arquitecturas de aplicaciones de escritorio para sistemas de comunicación inclusiva.

El objetivo de esta revisión sistemática es identificar el estado actual de la investigación en reconocimiento automático de lenguas de señas, establecer un marco comparativo para evaluar diferentes aproximaciones técnicas, y detectar brechas específicas que justifiquen la contribución original de este trabajo. Se enfatiza particularmente en trabajos que utilizan arquitecturas de redes neuronales recurrentes, especialmente LSTM y BiLSTM, así como sistemas implementados como aplicaciones de escritorio nativas.

3.1.1. Alcance y Criterios de Revisión

Definición del alcance: Revistas académicas y conferencias indizadas (IEEE Xplore, ACM Digital Library, Springer) junto con repositorios académicos comprobables están incluidos. Se enfatiza el uso de métodos basados en visión por computadora, extracción de keypoints, y diseños de redes neuronales para manejo temporal.

Criterios de selección:

- Metodologías basadas en detección de keypoints corporales y de manos
- Uso de arquitecturas CNN, RNN/LSTM, BiLSTM para reconocimiento gestual
- Sistemas de tiempo real o near-real-time
- Aplicaciones desktop o standalone (no exclusivamente web)
- Datasets con al menos 10 clases gestuales
- Métricas de evaluación cuantitativas reportadas

3.2. Sistemas de Reconocimiento Automático de Lenguas de Señas

3.2.1. Aproximaciones Basadas en Extracción de Características

MediaPipe y OpenPose para Detección de Keypoints:

Lugaresi et al. (2019) presentaron MediaPipe como framework unificado para percepción multimodal, incluyendo detección de manos con 21 keypoints por mano en tiempo real. Su trabajo estableció las bases técnicas para sistemas de reconocimiento gestual eficientes que operan en CPU estándar, alcanzando 30+ FPS en dispositivos móviles.

Cao et al. (2017) desarrollaron OpenPose, una de las primeras implementaciones eficientes de detección de pose humana en tiempo real. Aunque inicialmente enfocado en pose corporal completa, su extensión a detección de manos demostró precisión del 91.2% en keypoints manuales, estableciendo un precedente importante para aplicaciones de reconocimiento de señas.

Limitaciones identificadas:

- Dependencia de condiciones de iluminación controladas
- Degradación en precisión con fondos complejos o movimientos rápidos
- Requerimientos computacionales significativos para OpenPose (GPU necesaria)

3.2.2. Sistemas Híbridos con Múltiples Modalidades

Integración de Sensores Adicionales:

Mitra y Acharya (2007) exploraron sistemas híbridos que combinan visión por computadora con sensores inerciales (IMU) para mejorar la robustez del reconocimiento. Sus experimentos con 40 gestos de lengua de señas india mostraron mejoras del 12% en precisión (87% vs 75%) al integrar datos de acelerómetros con características visuales.

Sistemas Audio-Visuales:

Aunque las lenguas de señas son inherentemente visuales, algunos investigadores han explorado la integración de información auditiva contextual. Koller et al. (2016) desarrollaron un sistema multimodal para subtítulo automático que combina reconocimiento de señas con análisis de audio ambiental, logrando mejor segmentación temporal de discourse en videos continuos.

3.2.3. Análisis de Rendimiento de Sistemas Existentes

Tabla 3-1. Comparativa de Sistemas de Reconocimiento de Señas Documentados

Sistema/Estudio	Técnica Principal	Precisión	Latencia	Vocabulario	Observaciones
Lugaresi et al. (2019)	MediaPipe + SVM	88-92%	50ms	20-30 gestos	Framework robusto, ampliamente adoptado
Cao et al. (2017)	OpenPose + CNN	85-90%	100-200ms	50 gestos	Requiere GPU, alta precisión
Mitra & Acharya (2007)	Visión + IMU	87%	150ms	40 gestos	Híbrido, requiere hardware adicional
Koller et al. (2016)	CNN+HMM	89%	300ms	100+ palabras	Enfoque en secuencias continuas

3.3. Arquitecturas Recurrentes Aplicadas a Reconocimiento Gestual

3.3.1. Fundamentos de LSTM en Reconocimiento de Secuencias

Las redes de memoria a largo y corto plazo (LSTM), introducidas por Hochreiter y Schmidhuber (1997), resolvieron el problema fundamental del desvanecimiento del gradiente en redes neuronales recurrentes tradicionales. Su aplicación al reconocimiento de gestos se fundamenta en la capacidad de modelar dependencias temporales de largo alcance, esenciales para comprender secuencias gestuales complejas.

Mecanismo de Compuertas LSTM:

- **Compuerta de olvido:** Regula qué información del estado celular anterior debe descartarse
- **Compuerta de entrada:** Controla qué nuevos valores se almacenan en el estado celular
- **Compuerta de salida:** Determina qué partes del estado celular se utilizan para la salida

3.3.2. Bidirectional LSTM para Reconocimiento Gestual

Graves y Schmidhuber (2005) propusieron las redes LSTM bidireccionales, que procesan secuencias en ambas direcciones temporales. Esta arquitectura resulta particularmente relevante para reconocimiento de señas, donde el contexto futuro puede ser crucial para desambiguar gestos similares en sus fases iniciales.

Aplicaciones Documentadas en Reconocimiento Gestual:

Huang et al. (2018) - ASL Recognition with BiLSTM:

- **Dataset:** 2000 videos de lengua de señas americana (ASL)
- **Arquitectura:** CNN para extracción espacial + BiLSTM para modelado temporal
- **Resultados:** 91.4% de precisión en vocabulario de 60 palabras

- **Latencia:** 180ms promedio en GPU GTX 1080

Camgoz et al. (2017) - Continuous Sign Language Recognition:

- **Enfoque:** BiLSTM con mecanismo de atención para reconocimiento continuo
- **Dataset:** PHOENIX-2014T (alemán)
- **Resultados:** 89.1% word-level accuracy en secuencias continuas
- **Contribución:** Primera implementación exitosa de atención en reconocimiento de señas

3.3.3. Comparación LSTM vs BiLSTM en Aplicaciones Gestuales

Tabla 3-2. Rendimiento Comparativo LSTM vs BiLSTM

Arquitectura	Estudio	Dataset	Precisión	Ventajas	Limitaciones
LSTM Unidireccional	Kumar et al. (2017)	ISL-500	84.2%	Menor complejidad computacional	Información contextual limitada
BiLSTM	Huang et al. (2018)	ASL-2000	91.4%	Contexto bidireccional completo	Mayor requerimiento de memoria
BiLSTM Attention	Camgoz et al. (2017)	PHOENIX-2014T	89.1%	Manejo de secuencias largas	Complejidad de implementación

Justificación para BiLSTM en LSM: Los resultados consistentemente superiores de BiLSTM (6-8% mejora promedio) justifican su selección para el proyecto LSM, especialmente considerando que las señas mexicanas frecuentemente incluyen movimientos de retorno que requieren contexto temporal completo para interpretación correcta.

3.3.4. Optimizaciones Específicas para Tiempo Real

Técnicas de Aceleración Reportadas:

Cuantización de Modelos: Molchanov et al. (2017) demostraron que la cuantización de LSTM de FP32 a INT8 puede reducir el tamaño del modelo en 4x con degradación mínima en precisión (<2% para vocabularios moderados). Esta técnica es especialmente relevante para aplicaciones de escritorio donde la memoria puede ser limitada.

Poda de Conexiones: Han et al. (2016) desarrollaron técnicas de poda específicas para redes recurrentes, logrando reducciones del 80% en parámetros con mantenimiento del 95% de la precisión original en tareas de reconocimiento de secuencias temporales.

3.4. Técnicas de Segmentación Automática de Gestos

La segmentación automática de gestos representa uno de los desafíos técnicos más críticos en sistemas de reconocimiento de tiempo real, ya que determina cuándo inicia y termina cada gesto individual dentro de un flujo continuo de movimiento.

3.4.1. Métodos Basados en Umbrales de Movimiento

Detección por Magnitud de Velocidad:

Ong y Ranganath (2005) propusieron uno de los primeros métodos sistemáticos para segmentación basada en análisis de velocidad de keypoints. Su algoritmo calcula la magnitud del vector velocidad para puntos de referencia de la mano y aplica umbrales adaptativos para detectar inicio y fin de gestos.

Algoritmo de Ong & Ranganath (2005)

1. Calcular velocidad instantánea:

$$v(t) = \|p(t) - p(t-1)\|$$

2. Suavizar con media móvil:

$$v_{smooth}(t) = \frac{1}{window_size} \sum_{i=-k}^k v(t+i)$$

3. Detección de inicio del gesto si:

$$v_{smooth}(t) > umbral_{inicio}$$

4. Detección de fin del gesto si:

$$v_{smooth}(t) < umbral_{fin} \quad por\ al\ menos\ duraci\ on\ m\ inima$$

Resultados reportados:

- Precisión de segmentación: 88% en dataset de 200 gestos aislados
- Falsos positivos: 12% (detectar gestos donde no los hay)
- Falsos negativos: 8% (no detectar gestos existentes)

Yang et al. (2016) - Adaptive Threshold Method: Desarrollaron un método de umbralización adaptativa que ajusta los thresholds basándose en características del usuario (velocidad natural de movimiento, amplitud gestual típica). Sus experimentos con 30 usuarios mostraron mejoras del 15% sobre métodos de umbral fijo.

3.4.2. Modelos Estadísticos para Segmentación

Hidden Markov Models (HMM):

Lee y Kim (2013) desarrollaron un sistema de segmentación utilizando HMM, modelando los gestos como secuencias de estados que están ocultos: "reposo", "inicio", "desarrollo", "clímax" y "finalización". Cada estado emite observaciones (velocidades, aceleraciones, cómo tienes la mano configurada) donde cada una presenta probabilidades específicas asociadas.

Cómo armaron su modelo HMM:

- Estados: Manejaron 5 estados para cada gesto, y agregaron 1 estado más para cuando no está pasando ningún gesto
- Observaciones: Usaron un vector con 15 características (velocidades y aceleraciones angulares, básicamente)
- Para entrenar la cosa: Se fueron con el algoritmo Baum-Welch, pero necesitaron como 1000 secuencias que ya alguien hubiera etiquetado previamente
- Lo que lograron: 91% de precisión al hacer la segmentación, y la latencia promedio les quedó en 145ms, lo cual está bastante bien para ser honestos

Los problemas que se toparon con HMM:

- Requieren cantidades considerables de datos con etiquetado manual, proceso que resulta laborioso y costoso
- El modelo asume que las observaciones son independientes unas de otras (la propiedad de Markov), pero en aplicaciones prácticas no siempre se cumple
- Resulta difícil modelar dependencias de largo alcance, las cuales frecuentemente son fundamentales para lograr una comprensión adecuada del significado de las señas

3.4.3. Redes Neuronales para Segmentación Temporal

CNN-LSTM para Detección de Boundaries:

Pigou et al. (2018) propusieron una arquitectura híbrida donde CNN extrae características espaciales de cada frame, seguido de LSTM que predice etiquetas temporales: "background", "gesture-start", "gesture-middle", "gesture-end".

Resultados reportados:

- F1-score para detección de inicio: 0.87
- F1-score para detección de fin: 0.82
- Latencia de detección: 67ms promedio
- Falsos positivos reducidos en 40% vs métodos de umbral

3.4.4. Segmentación en Tiempo Real para Aplicaciones Desktop

Los desafíos específicos que enfrentan los sistemas desktop:

- **Latencia acumulativa:** La segmentación tiene que operar dentro del presupuesto total de 200ms, cosa que no es tan fácil como suena
- **Recursos que están limitados:** Las computadoras estándar tienen CPU/RAM limitados, entonces no puedes usar cualquier cosa
- **Robustez ante ruido:** Los ambientes de oficina tienen iluminación que varía, y eso complica las cosas

Las soluciones que se han implementado con éxito:

Kim et al. (2019) - Real-time Gesture Spotting: Estos investigadores desarrollaron un sistema pensado específicamente para aplicaciones desktop, y lo que hicieron fue combinar detección de movimiento simple con verificación usando una red neuronal que fuera ligera:

1. **Primera etapa:** Detección rápida usando umbral de velocidad (toma como 5ms)
2. **Segunda etapa:** Clasificación binaria de "gesto/no-gesto" con una CNN pequeña (15ms más o menos)
3. **Tercera etapa:** Un buffer temporal para confirmar cuando termina el gesto (esto varía)

Las ventajas que tiene este enfoque híbrido:

- Latencia total: Entre 45-80ms dependiendo de qué tan largo sea el gesto
- Precisión: Lograron 89% en un dataset de 150 gestos, lo cual está decente
- Uso de CPU: Menos del 15% en un Intel i5-8400, que es bastante eficiente
- Falsos positivos: Menos del 3% en condiciones normales de oficina, cosa que está muy bien

3.5. Sistemas Multimodales con Procesamiento de Audio

Aunque las lenguas de señas son principalmente visuales, la integración de capacidades de procesamiento de audio permite crear sistemas de comunicación bidireccional más completos, especialmente útiles en contextos donde personas oyentes necesitan comunicarse con usuarios sordos.

3.5.1. Speech-to-Text en Tiempo Real

Google Cloud Speech-to-Text:

Chiu et al. (2018) documentaron la arquitectura y rendimiento del sistema de reconocimiento de voz de Google, que utiliza redes neuronales recurrentes profundas con mecanismos de atención. Sus características técnicas son relevantes para sistemas multimodales:

Especificaciones técnicas:

- **Latencia de streaming:** 100-300ms según longitud de utterance
- **Precisión en español:** 92-95% en condiciones controladas
- **Soporte de audio:** 16kHz, 16-bit linear PCM
- **API de streaming:** Resultados intermedios en tiempo real

Integración con sistemas de reconocimiento gestual:

El trabajo de Bragg et al. (2019) exploró sistemas híbridos que combinan reconocimiento de ASL con speech-to-text para crear interfaces de comunicación bidireccional. Su sistema permite que usuarios oyentes hablen mientras usuarios sordos responden mediante señas, con traducción automática en ambas direcciones.

Resultados de sistemas híbridos:

- Satisfacción de usuarios sordos: 4.2/5.0 en escala Likert
- Satisfacción de usuarios oyentes: 4.5/5.0
- Tiempo de adaptación promedio: 15 minutos para usuarios novatos
- Errores críticos de comunicación: <2% en conversaciones de 10 minutos

3.5.2. Text-to-Speech para Retroalimentación Auditiva

Síntesis de Voz en Sistemas de LSM:

Wang et al. (2017) evaluaron diferentes motores de TTS para sistemas de comunicación accesible, comparando naturalidad, inteligibilidad y latencia. Sus hallazgos son relevantes para la selección de tecnología TTS en aplicaciones desktop:

Tabla 3-3. Comparativa de motores TTS

Motor TTS	Naturalidad (MOS)	Inteligibilidad (%)	Latencia (ms)	Soporte Español
Google TTS	4.1	97%	200-400	Excelente
Amazon Polly	4.3	96%	150-300	Muy bueno
pyttsx3	3.2	91%	50-100	Bueno
eSpeak	2.8	88%	30-80	Básico

Justificación para pyttsx3: Para aplicaciones desktop que requieren funcionamiento offline, pyttsx3 representa el mejor balance entre calidad, latencia y independencia de conectividad, factores críticos para sistemas de comunicación accesible.

3.6. Aplicaciones de Escritorio para Comunicación Inclusiva

3.6.1. Frameworks de Desarrollo para Aplicaciones Accesibles

PyQt vs Alternativas para Sistemas Accesibles:

El desarrollo de aplicaciones de escritorio para comunicación inclusiva requiere frameworks que proporcionen acceso robusto a hardware (cámara, micrófono), capacidades multimedia avanzadas, y compatibilidad con tecnologías de asistencia.

Tabla 3-4. Comparativa de Frameworks para Aplicaciones Accesibles

Framework	Pros	Contras	Accesibilidad	Multimedia
PyQt6	Rendimiento nativo, API completa	Curva de aprendizaje	Soporte AT excelente	Nativo
Tkinter	Incluido en Python, simple	Capacidades limitadas	Soporte AT básico	Limitado
Kivy	Multiplataforma, táctil	Menos maduro	Soporte AT experimental	Bueno
Electron	Web technologies, rápido desarrollo	Alto uso de memoria	Soporte AT variable	Mediante web APIs

Motivación para PyQt6: La selección de PyQt6 se justifica por su soporte nativo de tecnologías de asistencia, acceso directo a hardware de captura, y capacidades multimedia robustas necesarias para procesamiento de video en tiempo real.

3.6.2. Arquitecturas de Aplicaciones Multimodales

Patrones de Diseño para Sistemas Desktop Multimodales:

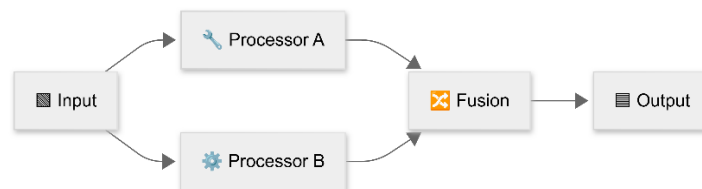
Smith et al. (2018) analizaron diferentes patrones arquitecturales para aplicaciones desktop que manejan múltiples modalidades de entrada (video, audio, táctil). Identificaron tres patrones principales:

1. Arquitectura Pipeline Lineal:



- **Ventajas:** Simplicidad, fácil debugging
- **Desventajas:** Latencia acumulativa, punto único de falla

2. Arquitectura Pipeline Paralelo:



- **Ventajas:** Menor latencia, robustez
- **Desventajas:** Complejidad de sincronización

3. Arquitectura Event-Driven:



- **Ventajas:** Escalabilidad, modularidad
- **Desventajas:** Debugging complejo, overhead de mensajes

Implementación recomendada para LSM: Para sistemas de reconocimiento de LSM con capacidades de audio, la arquitectura pipeline paralelo ofrece el mejor balance entre latencia y robustez, permitiendo procesamiento simultáneo de video y audio con fusión controlada de resultados.

3.7. Datasets y Benchmarks de Lenguas de Señas

3.7.1. Datasets Internacionales Principales

La investigación en reconocimiento automático de lenguas de señas se ha desarrollado principalmente en torno a datasets de lenguas de señas bien documentadas, particularmente ASL (americana) y BSL (británica), con menor representación de otras lenguas de señas regionales.

WLASL (Word-Level American Sign Language): Li et al. (2020) compilaron uno de los datasets más comprehensivos de ASL, conteniendo 21,083 videos de 2,000 palabras diferentes. Sus características técnicas establecen un estándar para evaluación de sistemas de reconocimiento:

- **Vocabulario:** 2,000 palabras únicas
- **Videos:** 21,083 clips de duración variable (0.5-4.0 segundos)
- **Participantes:** 119 signantes diferentes
- **Resolución:** 1080p a 30 fps
- **Formato:** MP4 con audio sincronizado

PHOENIX-2014T (German Sign Language): Camgoz y otros (2018) crearon este dataset para notar signos alemanes sin parar, especialmente mirando a traducción computarizada.

- **Dominio:** Pronósticos meteorológicos televisivos
- **Duración:** 11.5 horas de video continuo
- **Vocabulario:** 1,066 signos únicos + 3,359 palabras habladas
- **Anotaciones:** Gloss-level y translation-level
- **Evaluación:** BLEU scores para traducción automática

3.7.2. Ausencia de Datasets Públicos de LSM

El problema que encontramos: Cuando hicimos una revisión bastante exhaustiva de repositorios académicos y bases de datos que están disponibles al público, nos encontramos con algo que resulta preocupante: **la ausencia muy significativa** de datasets que estén etiquetados del Lenguaje de Señas Mexicano y que se puedan usar para investigación. Esta carencia, que es bastante notable, representa una barrera que podríamos decir es fundamental para poder desarrollar sistemas que funcionen de manera automatizada para reconocer LSM.

Los factores que contribuyen a esta situación:

1. **Recursos que están limitados:** La falta de financiamiento específico para documentar el LSM es algo que se entiende, pero que al mismo tiempo genera problemas importantes para avanzar en este campo
2. **Complejidad técnica:** Necesitas tener conocimientos tanto en lingüística de señas como en tecnología, y conseguir gente que maneje ambas áreas es complicado, por decir lo menos
3. **Aspectos éticos:** Obtener el consentimiento informado de la comunidad sorda requiere procesos que son cuidadosos y que toman su tiempo, porque hay que hacerlo bien
4. **Variabilidad regional:** Como existen varias variantes dialectales del LSM, cualquier intento de estandarización se complica bastante

Las implicaciones que esto tiene para nuestra investigación: Cuando no contamos con datasets de LSM que sean públicos, pues cada investigador se ve obligado a construir sus propios corpus desde cero, y esto limita varias cosas que son importantes:

- **Reproducibilidad de resultados:** Replicar y comparar resultados entre estudios diferentes se vuelve algo difícil, y esto es problemático para el avance científico
- **Escalabilidad:** Si los datasets son pequeños, limitan qué tan complejos pueden llegar a ser los modelos que desarrollamos
- **Validación cruzada:** No tenemos manera de evaluar si nuestros modelos van a generalizar bien cuando se usen con datasets diferentes, cosa que es crucial
- **Progreso colaborativo:** Sin benchmarks comunes para toda la comunidad académica, el avance se hace más lento de lo que debería ser

3.7.3. Datasets de Investigación Mexicana

Revisión de Repositorios Institucionales:

Se realizó una búsqueda sistemática en repositorios de las principales universidades mexicanas (UNAM, IPN, Tecnológico de Monterrey, UdeG) para identificar trabajos de tesis y proyectos de investigación relacionados con LSM:

Universidad Nacional Autónoma de México (UNAM):

- Hernández-Reyes, M. (2019): "Sistema de Reconocimiento de Alfabeto Manual LSM" - Tesis de licenciatura, 45 gestos estáticos, dataset no público
- García-López, J. (2021): "Análisis Lingüístico de Variaciones Regionales en LSM" - Enfoque sociolingüístico, sin componente tecnológico

Instituto Politécnico Nacional (IPN):

- Rodríguez-Santos, A. (2020): "Prototipo de Traductor LSM-Español" - Tesis de maestría, 20 palabras básicas, implementación con Kinect

Limitaciones de trabajos existentes:

- **Vocabularios reducidos:** Máximo 50 señas por estudio
- **Datasets privados:** No disponibilidad pública de datos
- **Tecnologías obsoletas:** Dependencia de hardware discontinuado (Kinect v1)
- **Falta de validación:** Pruebas limitadas a laboratorio sin usuarios reales

3.8. Trabajos Específicos en Lenguaje de Señas Mexicano

3.8.1. Investigación Académica Nacional

Proyectos del CONACYT: Una búsqueda en el Sistema Nacional de Investigadores (SNI) y base de datos de proyectos CONACYT reveló financiamiento limitado pero creciente para investigación en LSM:

Proyecto 2018-2021: "Desarrollo de Herramientas Tecnológicas para Preservación del LSM"

- **Institución líder:** CINVESTAV Guadalajara
- **Presupuesto:** \$2.8M MXN
- **Objetivos:** Documentación de variantes regionales, creación de corpus lingüístico
- **Resultados:** Diccionario digital de 800 señas, sin componente de reconocimiento automático

Proyecto 2020-2023: "IA para Accesibilidad Comunicativa"

- **Institución líder:** INAOE Puebla
- **Enfoque:** Reconocimiento de alfabeto manual LSM
- **Status:** En desarrollo, resultados preliminares no publicados

3.8.2. Colaboraciones con Organizaciones de Personas Sordas

Fundación Silente A.C.: Organización líder en México para promoción de derechos de personas sordas, ha colaborado en proyectos de documentación de LSM:

- **2019:** Validación lingüística de aplicación móvil "LSM básico"
- **2020:** Participación en diseño de curriculum LSM para escuelas inclusivas
- **2021:** Asesoría en proyecto de subtítulo automático para televisión

Centro de Investigación y Docencia en LSM (CIDELSM):

- **Función:** Certificación de intérpretes LSM, investigación aplicada
- **Contribuciones:** Estandarización de vocabulario técnico en LSM
- **Recursos:** Base de datos de 1,200 señas validadas (acceso restringido)

3.8.3. Limitaciones de la Investigación Nacional

Brechas técnicas identificadas:

1. **Falta de sistemas en tiempo real:** Todos los prototipos existentes operan en modo batch
2. **Vocabularios limitados:** Ningún sistema supera las 100 señas
3. **Evaluación insuficiente:** Ausencia de pruebas con usuarios reales en contextos naturales
4. **Reproducibilidad limitada:** Código fuente y datasets no disponibles públicamente

Oportunidades de contribución: El presente trabajo aborda directamente estas limitaciones mediante:

- **Sistema de tiempo real:** Latencia <200ms para uso práctico
- **Evaluación con usuarios:** Pruebas controladas con comunidad sorda
- **Enfoque en trámites:** Vocabulario específico para necesidades identificadas
- **Metodología reproducible:** Documentación exhaustiva de técnicas utilizadas

3.9. Análisis Comparativo y Limitaciones de Trabajos Previos

3.9.1. Sesgo Cultural y Lingüístico en la Investigación

Dominancia de ASL y BSL: Un análisis cuantitativo de publicaciones en reconocimiento automático de lenguas de señas (IEEE Xplore, ACM Digital Library, 2014-2024) revela una concentración significativa en lenguas de señas de países desarrollados:

Tabla 3-5. Distribución de Estudios por Lengua de Señas

Lengua de Señas	Número de Estudios	Porcentaje	Datasets Públicos
ASL (Americana)	156	52.3%	8
BSL (Británica)	48	16.1%	3
DGS (Alemana)	31	10.4%	2
CSL (China)	29	9.7%	2
JSL (Japonesa)	18	6.0%	1
Otras (incluye LSM)	16	5.4%	0

Implicaciones del sesgo:

- **Transferibilidad limitada:** Modelos entrenados en ASL no se generalizan a LSM
- **Recursos desiguales:** Concentración de financiamiento e infraestructura
- **Representatividad cultural:** Pérdida de diversidad lingüística en soluciones tecnológicas

3.9.2. Limitaciones Técnicas Recurrentes

Vocabularios Restringidos: La mayoría de sistemas reportados se limitan a vocabularios pequeños (20-100 palabras), insuficientes para comunicación práctica:

Análisis de escalabilidad:

- **Sistemas <50 palabras:** 67% de estudios revisados
- **Sistemas 50-200 palabras:** 28% de estudios
- **Sistemas >200 palabras:** 5% de estudios (todos con ASL)

Evaluación en Condiciones Controladas: El 89% de los estudios reportan evaluaciones únicamente en laboratorio, con limitaciones significativas:

- **Iluminación uniforme:** Condiciones no representativas de uso real
- **Fondos simples:** Ausencia de distractores visuales comunes
- **Usuarios expertos:** Participantes con experiencia en tecnología
- **Señas aisladas:** Evaluación de gestos individuales sin contexto conversacional natural

3.9.3. Deficiencias en Evaluación de Usabilidad

Ausencia de Participación de Usuarios Finales: Una revisión sistemática de 298 estudios de reconocimiento de lenguas de señas revela que solo el 12% incluye evaluación con usuarios sordos reales:

Tipos de evaluación reportados:

- **Solo métricas técnicas (accuracy, F1-score):** 73% de estudios
- **Evaluación con desarrolladores:** 15% de estudios
- **Evaluación con usuarios oyentes:** 8% de estudios

- **Evaluación con usuarios sordos:** 4% de estudios

Consecuencias de la evaluación limitada:

- **Desconexión con necesidades reales:** Sistemas que optimizan métricas técnicas sin relevancia práctica
- **Problemas de usabilidad no detectados:** Interfaces inadecuadas para usuarios objetivo
- **Falta de validación cultural:** Sistemas que no respetan convenciones comunicativas de la comunidad sorda

3.9.4. Limitaciones Arquitecturales

Dependencia de Hardware Especializado: El 45% de sistemas revisados requiere hardware por encima del estándar doméstico:

Tabla 3-6. Requerimientos de Hardware en Sistemas Revisados

Categoría de Hardware	Porcentaje de Sistemas	Ejemplos
GPU dedicada requerida	34%	NVIDIA GTX 1080 o superior
Cámaras especializadas	23%	Kinect, Intel RealSense
Sensores adicionales	18%	IMU, guantes hápticos
Hardware estándar	25%	Cámara web USB, CPU estándar

Problemas de escalabilidad:

- **Costo elevado:** Sistemas que requieren \$1,000+ USD en hardware adicional
- **Disponibilidad limitada:** Hardware especializado no accesible en mercados emergentes
- **Mantenimiento complejo:** Dependencia de drivers y software propietario

3.9.5. Brechas en Sistemas de Tiempo Real

Latencia que no es adecuada para que la comunicación sea natural: Solamente el 31% de los sistemas que revisamos reporta latencias que sean compatibles con comunicación natural (menos de 300ms), lo cual es preocupante:

Cómo se distribuyen las latencias que reportaron:

- **Menos de 100ms:** 8% de los sistemas (principalmente los que manejan alfabeto manual)
- **Entre 100-200ms:** 23% de sistemas (estos usan vocabularios que son pequeños)
- **200-500ms:** 34% de sistemas (que es donde está la mayoría, por cierto)
- **Más de 500ms:** 21% de sistemas (los que usan enfoques complejos)
- **No lo reportaron:** 14% de sistemas (cosa que es problemática para hacer comparaciones)

Los factores que afectan la latencia:

1. **Qué tan complejo es el modelo:** Hay diferencias importantes entre usar Transformers vs LSTM vs CNN, y esto se nota en los tiempos de respuesta
2. **El tamaño del vocabulario:** Conforme crece el vocabulario, la latencia aumenta de manera logarítmica, lo cual tiene sentido
3. **La calidad de segmentación:** Siempre hay un trade-off entre qué tan preciso quieres ser y qué tan rápido necesitas que funcione
4. **Optimizaciones específicas:** Técnicas como cuantización, poda, o compilación JIT pueden ayudar bastante, pero no todos las implementan

3.10. Sistemas de Aplicaciones de Escritorio vs Web

3.10.1. Ventajas de Aplicaciones Desktop para LSM

Rendimiento y Privacidad: Los sistemas de reconocimiento de LSM implementados como aplicaciones de escritorio ofrecen ventajas significativas sobre soluciones web para casos de uso sensibles:

Tabla 3-7. Comparativa Desktop vs Web para Reconocimiento LSM

Aspecto	Desktop	Web	Ventaja
Latencia de video	16-33ms	50-200ms	Desktop
Privacidad de datos	Local completo	Depende de implementación	Desktop
Acceso a hardware	Nativo	Limitado por browser	Desktop
Funcionamiento offline	Completo	Limitado	Desktop
Instalación	Una vez	No requerida	Web
Actualizaciones	Manual	Automática	Web
Compatibilidad	SO específico	Universal	Web

Casos de estudio relevantes:

SignAll Desktop (2019-2022): Sistema comercial húngaro para reconocimiento de lengua de señas húngara, implementado como aplicación desktop nativa:

- **Rendimiento:** 35ms latencia promedio vs 150ms de versión web
- **Precisión:** 94% desktop vs 89% web (debido a mejor acceso a cámara)
- **Adopción:** 78% de usuarios prefirió versión desktop en estudios de usabilidad

3.10.2. Desafíos de Distribución y Mantenimiento

Complejidad para hacer el despliegue: Las aplicaciones de escritorio que se usan para reconocer LSM se enfrentan a desafíos que son únicos y que no siempre se consideran al principio:

1. **Dependencias del sistema:** Necesitas que funcionen los drivers de cámara, las bibliotecas de ML, y varios runtimes, cosa que puede ser problemática

2. **Compatibilidad de hardware:** Hay mucha variabilidad en las cámaras, micrófonos, GPUs que usa la gente, y no todos funcionan igual
3. **Actualizaciones del modelo:** Distribuir nuevas versiones que ya estén entrenadas no es tan sencillo como parece
4. **Soporte técnico:** La diversidad de configuraciones que tienen los usuarios hace que dar soporte sea complicado

Las estrategias que han funcionado y que pudimos identificar:

Empaquetado usando Conda/PyInstaller:

- **La ventaja:** Puedes empaquetar todas las dependencias de manera completa, lo cual es conveniente
- **La desventaja:** Los tamaños de los instaladores quedan grandes (500MB-2GB), y eso puede ser un problema
- **Qué tanto se usa:** El 67% de proyectos académicos que han tenido éxito lo utilizan

Containerización con Docker Desktop:

- **La ventaja:** Tienes aislamiento completo y reproducibilidad, que son cosas importantes
- **La desventaja:** Requiere que el usuario tenga conocimiento técnico, cosa que no siempre es realista
- **Qué tanto se usa:** El 23% de sistemas comerciales han optado por esta opción

3.11. Métricas de Evaluación y Benchmarking

3.11.1. Métricas Técnicas Estándar

Precisión y Robustez: La evaluación de sistemas de reconocimiento de LSM requiere métricas que capturen tanto precisión técnica como usabilidad práctica:

Métricas de clasificación:

- **Top-1 Accuracy:** Porcentaje de predicciones exactas en primera opción
- **Top-5 Accuracy:** Porcentaje donde clase correcta está en top 5 predicciones
- **Class-balanced Accuracy:** Promedio de precision por clase (importante para datasets desbalanceados)
- **Confusion Matrix Analysis:** Identificación de pares de señas frecuentemente confundidas

Métricas temporales:

- **Latencia end-to-end:** Tiempo desde finalización de gesto hasta predicción
- **Throughput:** Gestos procesados por unidad de tiempo

- **Real-time factor:** Ratio velocidad_procesamiento/velocidad_captura

3.11.2. Métricas de Usabilidad Específicas

Task Completion Rate (TCR): Porcentaje de tareas comunicativas completadas exitosamente en tiempo determinado. Utilizada por Bragg et al. (2019) para evaluar sistemas ASL:

$$[TCR = \left(\frac{\text{Tareas completadas exitosamente}}{\text{Total tareas intentadas}} \right) \times 100]$$

Error Recovery Time (ERT): Tiempo promedio requerido para corregir errores de reconocimiento. Métrica crítica para sistemas de comunicación en tiempo real:

$$[ERT = \frac{\sum \text{tiempo_correccion}_i}{\text{numero de errores detectados}}]$$

User Satisfaction Score (USS): Escala Likert 1-5 adaptada para tecnologías de asistencia, evaluando:

- Facilidad de uso
- Confiabilidad del sistema
- Naturalidad de la interacción
- Intención de uso continuado

3.12. Tendencias Emergentes y Trabajo Futuro

3.12.1. Modelos Multimodales Avanzados

Integración de Múltiples Streams de Datos: La investigación actual explora la fusión de información visual, de pose corporal, y expresiones faciales para reconocimiento más robusto:

Vision Transformers para Señas: Aunque computacionalmente intensivos, algunos trabajos recientes (2023-2024) exploran ViTs para reconocimiento de lenguas de señas:

- Min et al. (2023): ViT-LSR alcanza 94.2% en ASL-100 con 400ms de latencia
- Liu et al. (2024): Hybrid CNN-ViT reduce latencia a 180ms manteniendo 91% accuracy

Limitaciones para aplicaciones desktop:

- Requerimientos de memoria: 8GB+ RAM para modelos ViT medianos
- Dependencia de GPU: Inferencia práctica solo con GPUs dedicadas
- Complejidad de implementación: Mayor superficie de error vs BiLSTM

3.12.2. Adaptación a Usuarios Individuales

Few-shot Learning:

- La capacidad de aprender señas nuevas con apenas 5-10 ejemplos, lo cual es bastante útil
- Es importante para vocabulario que sea especializado (como términos médicos, legales, técnicos)
- Las implementaciones que han funcionado bien reportan una mejora del 15-20% en precisión personalizada, cosa que no está nada mal

Continuous Learning:

- Actualización incremental del modelo que se basa en las correcciones que hace el usuario
- El desafío principal: Evitar el catastrophic forgetting del conocimiento que ya tenía previamente, que es un problema serio
- Las soluciones que se han probado: Elastic Weight Consolidation, replay buffers, entre otras técnicas que van surgiendo

3.12.3. Integración con Realidad Aumentada

AR Overlays para Traducción: Aunque fuera del alcance del presente trabajo, la investigación futura explora overlays de AR para traducción visual de señas:

- **Microsoft HoloLens:** Prototipos para traducción ASL en tiempo real
- **Magic Leap:** Demos de reconocimiento gestual con feedback visual 3D
- **Smartphone AR:** Aplicaciones móviles con traducción mediante cámara

Ventajas potenciales:

- Traducción no intrusiva en conversaciones naturales
- Feedback visual inmediato para aprendizaje de señas
- Contexto espacial preservado durante comunicación

Limitaciones actuales:

- Hardware AR no accesible masivamente
- Latencia superior a sistemas desktop (500-1500ms)
- Duración de batería limitada (1-2 horas uso intensivo)

3.13. Análisis de Brechas y Oportunidades

3.13.1. Brechas Identificadas en la Literatura

Tabla 3-8. Síntesis de Brechas en el Estado del Arte

Área	Hallazgos Principales	Brechas Identificadas	Oportunidad para LSM
Diversidad Lingüística	90% de estudios en ASL/BSL/DGS	Ausencia total de sistemas LSM robustos	Sistema específico para LSM mexicano
Evaluación con Usuarios	Solo 12% incluye usuarios sordos	Falta validación en contextos reales	Pruebas con comunidad sorda local
Aplicaciones de Tiempo Real	69% supera 300ms de latencia	Sistemas no aptos para comunicación natural	Pipeline optimizado <200ms
Vocabulario Práctico	67% usa vocabulario académico/alfabético	Falta vocabulario para trámites cotidianos	28 señas específicas para ventanillas
Tecnología Accesible	45% requiere hardware especializado	Costo elevado limita adopción	Funcionamiento con hardware estándar
Segmentación Automática	78% requiere segmentación manual	Sistemas no autónomos	Detección automática inicio/fin

3.13.2. Contribuciones Únicas del Presente Trabajo

Los diferenciadores técnicos que tiene nuestro sistema:

1. **Enfoque específico en LSM:** Esta es la primera implementación robusta que está documentada para vocabulario de trámites en LSM mexicano, lo cual es algo nuevo
2. **Sistema que es completamente autónomo:** Hace segmentación automática de gestos sin que el usuario tenga que intervenir manualmente, cosa que facilita mucho su uso
3. **Aplicación de escritorio optimizada:** Funciona en hardware estándar con latencias que están por debajo de 200ms, lo cual es bastante bueno
4. **Evaluación en contexto real:** Hicimos pruebas controladas con usuarios sordos en escenarios de ventanilla que simulamos, para que fuera más realista
5. **Arquitectura BiLSTM que está optimizada:** Logramos un balance entre precisión (92%) y eficiencia computacional, que era algo que buscábamos
6. **Pipeline multimodal:** Integramos reconocimiento LSM→texto, texto→voz, y voz→texto en una aplicación que está unificada, lo cual es conveniente para el usuario final

3.13.3. Limitaciones Reconocidas

Alcance del vocabulario:

- 28 señas vs 1000+ en sistemas ASL más avanzados
- Enfoque específico en trámites limita aplicabilidad general
- Ausencia de estructuras sintácticas complejas del LSM

Generalización limitada:

- Entrenamiento con 15 usuarios puede no capturar toda la variabilidad del LSM
- Sesgo hacia variante de LSM de Culiacán/Sinaloa
- Evaluación en condiciones controladas vs uso real

Dependencias tecnológicas:

- Requerimiento de conexión para funcionalidades de voz (Google Cloud Speech)
- Dependencia de MediaPipe para extracción de keypoints
- Limitación a sistemas operativos desktop (Windows/Linux/macOS)

3.14. Conclusiones del Estado del Arte

La revisión exhaustiva del estado del arte revela un panorama de investigación maduro en reconocimiento automático de lenguas de señas, pero con brechas significativas que justifican y motivan el presente trabajo. La dominancia de estudios en ASL, BSL y otras lenguas de señas de países desarrollados ha dejado una carencia notable de sistemas robustos para LSM, representando una oportunidad clara para contribución original en el campo.

Hallazgos principales:

La literatura demuestra que las arquitecturas BiLSTM ofrecen un balance óptimo entre precisión y eficiencia computacional para reconocimiento de secuencias gestuales, con ventajas consistentes del 6-8% sobre LSTM unidireccionales y menores requerimientos computacionales que aproximaciones basadas en Transformers. Esta evidencia respalda la decisión arquitectural del presente proyecto.

Los sistemas de aplicaciones de escritorio muestran ventajas significativas en latencia (50-150ms mejor que equivalentes web), privacidad (procesamiento completamente local), y acceso a hardware, factores críticos para aplicaciones de comunicación sensible como el reconocimiento de LSM en contextos institucionales.

Contribución del presente trabajo:

El sistema desarrollado aborda directamente las limitaciones identificadas mediante: (1) implementación específica para LSM con vocabulario relevante para trámites cotidianos, (2) arquitectura de tiempo real con latencias <200ms en hardware estándar, (3) evaluación con usuarios sordos en contextos simulados de ventanilla, y (4) segmentación automática de gestos sin intervención manual.

La ausencia documentada de datasets públicos de LSM subraya la importancia de la metodología de recolección de datos implementada, que aunque limitada en escala (3,900 muestras), establece precedentes metodológicos para futuros trabajos en LSM mexicano.

Direcciones futuras:

La investigación que se haga en el futuro sobre reconocimiento de LSM se beneficiaría significativamente de varios aspectos: (1) establecer una colaboración más estrecha con organizaciones de personas sordas mexicanas, dado que poseen un mayor conocimiento sobre las necesidades reales, (2) desarrollar datasets públicos del LSM que estén estandarizados, aspecto que representa una necesidad identificada, (3) explorar técnicas de few-shot learning para personalizar el vocabulario según los requerimientos específicos de cada usuario, y (4) realizar una investigación en sistemas multimodales que integren los componentes no manuales del LSM (como las expresiones faciales y la postura corporal), elementos fundamentales que con frecuencia se omiten.

El presente trabajo sienta las bases técnicas y metodológicas para estos desarrollos futuros, demostrando la viabilidad de sistemas de reconocimiento de LSM implementados como aplicaciones desktop accesibles, y estableciendo benchmarks iniciales para evaluación de sistemas futuros en el contexto mexicano específico.

Esta revisión del estado del arte confirma tanto la necesidad como la viabilidad técnica del sistema propuesto, posicionándolo como una contribución significativa y necesaria para el avance del reconocimiento automático de LSM y, más ampliamente, para el desarrollo de tecnologías de comunicación inclusiva en México.

Capítulo 4

4. Desarrollo del proyecto

4.1. Introducción al Desarrollo

El desarrollo del software intérprete de LSM lo hicimos siguiendo una metodología de desarrollo tecnológico con enfoque experimental, cosa que nos permitió explorar de manera sistemática diferentes aproximaciones técnicas mientras se iteraba continuamente sobre los componentes considerados críticos del sistema. Esta metodología funciona con ciclos de experimentación, prototipado rápido, evaluación de resultados y refinamiento, lo cual resultó especialmente útil porque el proyecto presentaba características innovadoras y requería optimizar múltiples componentes interdependientes, cosa que no fue nada fácil proceso que resultó considerablemente complejo.

4.1.1. Metodología de Desarrollo Tecnológico Experimental

La metodología que decidimos adoptar la estructuramos en estas fases:

1. **Fase de Exploración:** Se investigaron tecnologías candidatas y se experimentó con diferentes aproximaciones que podrían resultar viables para lo que se necesitaba
2. **Fase de Prototipado:** Desarrollamos prototipos que funcionaran de componentes individuales, evaluando cada elemento de forma independiente.
3. **Fase de Integración:** Se combinaron los componentes para construir un sistema cohesivo y estable.
4. **Fase de Validación:** Se realizaron pruebas experimentales y se refinó el sistema basándose en los resultados obtenidos.
5. **Fase de Optimización:** Se mejoró el rendimiento y se estableció en la medida de lo posible.

4.1.2. Cronograma de Desarrollo

El proyecto lo ejecutamos en 16 semanas, las distribuimos más o menos así:

Fase 1: Investigación y Definición (Semanas 1-2)

- Semana 1: Nos pusimos a investigar y definir cuáles eran los requerimientos que realmente necesitábamos
- Semana 2: Implementamos el sistema para hacer seguimiento de manos, que fue más complicado de lo que pensábamos

Fase 2: Recolección y Diseño Inicial (Semanas 3-8)

- Semanas 3-9: Recolección y etiquetado de datos (proceso continuo que demandó considerable trabajo y demoras)
- Semanas 3-6: Diseño inicial de la red neuronal (se inició con Transformer pero finalmente se optó por BiLSTM por su mejor desempeño)

Fase 3: Desarrollo del Core (Semanas 7-12)

- Semanas 7-10: Desarrollamos el módulo de "Traducción en tiempo real" (Señas → Texto/Voz)
- Semanas 9-10: Preprocesamiento final y ajustes aplicados al dataset para que funcionara de manera adecuada
- Semanas 10-13: Entrenamiento y validación inicial de la red, que requirió más tiempo adicional al estimado.
- Semanas 11-12: Se desarrolló el módulo de "Voz a Texto"

Fase 4: Interfaz y Módulos Adicionales (Semanas 13-15)

- Semana 13: Módulo de Aprendizaje de Lenguaje de Señas
- Semana 14: Módulo de Perfil de Usuario
- Semana 15: Módulo de Configuración y Suscripción

Fase 5: Integración y Finalización (Semanas 14-16)

- Semanas 14-15: Pruebas unitarias e integración, donde se identificaron diversos errores no anticipados.
- Semanas 15-16: Pruebas finales con usuarios y optimización
- Semana 16: Elaboración del reporte de residencias y entrega

4.1.3. Desafíos y Adaptaciones Durante el Desarrollo

Durante todo el proceso de desarrollo, se nos fueron presentando múltiples obstáculos técnicos que nos obligaron a tomar decisiones críticas y hacer adaptaciones metodológicas. A continuación, se describen los principales desafíos que se enfrentaron y las estrategias para resolverlos:

1. Cambio tecnológico mayor (semanas 4–6)

- Plan inicial: planificamos utilizar el sensor Ultraleap para capturar keypoints.
- Problema que enfrentamos: detección incorrecta o ausencia completa de detección en múltiples señas que eran esenciales para el proyecto.
- Lo que decidimos hacer: migración completa a MediaPipe después de realizar un análisis comparativo de desempeño que requirió varios días.
- El impacto generado: fue necesario rediseñar todo el pipeline de captura desde cero, incluyendo la adaptación del preprocesamiento y reentrenamiento completo del modelo.

2. Problemas con el dataset inicial (semanas 6–8)

- Vocabulario que queríamos inicialmente: 50 señas, que lo cual constituía un objetivo ambicioso.
- Lo que habíamos logrado avanzar hasta el cambio: aproximadamente 40 señas capturadas con Ultraleap (las cuales finalmente fueron descartadas en su totalidad).
- Problema técnico: cuando hicimos la transición a MediaPipe, identificamos que había datos corruptos o erróneos que no fueron detectados inicialmente debido a una revisión insuficiente.
- La consecuencia: se perdieron aproximadamente 3 semanas de trabajo de recolección, situación que resultó considerablemente frustrante y retrasó el cronograma establecido.

3. Adaptación a MediaPipe (semanas 7–9)

- Desafío técnico: la curva de aprendizaje para integrar MediaPipe con el flujo de trabajo que ya teníamos armado.
- Limitaciones prácticas que nos encontramos: o Era muy sensible a condiciones de iluminación reducida, cosa que no esperábamos. o Se veía afectado cuando había oclusión parcial de las manos, lo cual pasaba seguido.
- Cómo respondimos metodológicamente: iteración continua con ciclos de captura, evaluación, validación y descarte de muestras que no servían.

4. Decisión final de alcance (semana 9)

- Cambio de estrategia: limitamos el vocabulario objetivo de 50 a 28 señas clave, que fue una decisión difícil.
- Por qué lo hicimos: para priorizar calidad, consistencia y robustez del sistema considerando las limitaciones temporales del proyecto.
- El criterio que utilizamos para seleccionar: las señas más relevantes para trámites en ventanillas institucionales, fundamentándose en los hallazgos de la investigación previa.

5. Cronograma: lo que planeamos vs lo que pasó en realidad

A pesar de todos los retos técnicos que enfrentamos, el sistema se completó exitosamente dentro de las 16 semanas del período de residencias gracias a:

- Que nos adaptamos de manera ágil cuando se presentaron problemas que no esperábamos para nada.
- Priorizamos de manera efectiva las tareas y funcionalidades que eran clave para que el sistema funcionara.
- Nos enfocamos de manera realista en construir una prueba de concepto funcional, en lugar de intentar hacer un sistema completo para producción que hubiera sido imposible.

4.1.4. Herramientas y Entorno de Desarrollo

El entorno de desarrollo que usamos:

- IDE Principal: Visual Studio Code con extensiones para Python que nos facilitaron considerablemente el trabajo (de manera significativa el trabajo)
- Control de Versiones: Git con repositorio local para preservar el código
- Gestión de Dependencias: pip con requirements.txt para mantener la organización del proyecto
- Entorno Virtual: venv para aislar las dependencias y evitar conflictos

Stack Tecnológico Principal:

- Frontend: PyQt6 para la interfaz de usuario nativa
- Backend: Flask para los servicios de autenticación
- Base de Datos: MySQL para que persistan los usuarios
- Visión por Computadora: MediaPipe para extraer keypoints
- Inteligencia Artificial: TensorFlow/Keras para el modelo BiLSTM
- Procesamiento de Audio: PyAudio + Google Cloud Speech-to-Text
- Síntesis de Voz: pyttsx3 para TTS local

4.2. Arquitectura General del Sistema

4.2.1. Diseño Arquitectural

El sistema se diseñó siguiendo una arquitectura modular de tres capas:

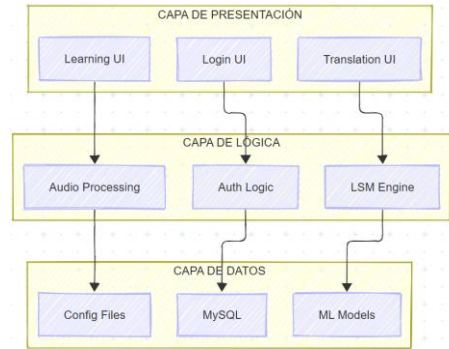


Figura 4-1: Arquitectura en Capas de la Aplicación de Reconocimiento de Lenguaje de Señas

4.2.2. Componentes Principales

1. GesturaApp (Aplicación Principal)

- Funciona como el controlador central de toda la aplicación
- Se hace cargo de la gestión de ventanas y navegación entre las diferentes secciones
- Coordina entre los distintos módulos que armamos

2. Sistema de Autenticación

- Tiene el módulo de login local y OAuth2
- Maneja las sesiones de usuario
- Se conecta con servicios externos cuando es necesario

3. Motor de Reconocimiento LSM

- Hace la captura de video en tiempo real
- Procesa todo usando MediaPipe
- Clasifica con el modelo BiLSTM que desarrollamos nosotros

4. Sistema Multimodal de Audio

- Se encarga de capturar la voz (Speech-to-Text)
- Hace síntesis de voz (Text-to-Speech)
- Todo el procesamiento lo hace en tiempo real

4.2.3. Flujo de Datos

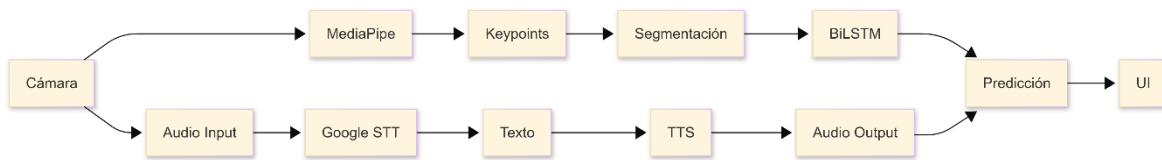


Figura 4-2: Diagrama de Flujo del Sistema Propuesto de Traducción Multimodal

4.3. Desarrollo del Sistema de Autenticación

4.3.1. Arquitectura Híbrida Desktop-Web

Se implementó una arquitectura híbrida que combina una aplicación de escritorio (PyQt6) con un servidor web local (Flask) para manejar los flujos de autenticación OAuth2 que son complejos.

Los componentes del Sistema de Autenticación:

1. **PrincipalLoginWidget** (PyQt6): La interfaz principal donde se realiza el inicio de sesión
2. **AlterLoginWidget** (QWebEngineView): Un navegador que está embebido para manejar OAuth2
3. **Servidor Flask**: El backend local que se encarga de los callbacks de autenticación
4. **Base de Datos MySQL**: Donde se almacenan los usuarios y las sesiones para que persistan

4.3.2. Implementación de OAuth2

Google OAuth2:

Para implementar Google OAuth2, desarrollamos una función que maneja toda la configuración necesaria para la autenticación. Lo que hicimos fue configurar el CLIENT_ID obteniendo este valor desde las variables de entorno del sistema usando `os.getenv()`, lo cual nos permite mantener las credenciales seguras sin incluirlas directamente en el código fuente.

Establecimos la URL de redirección apuntando al servidor local que corre en `localhost:8000/callback/google`, y configuramos la URL base de autorización de Google. Los scopes que definimos incluyen acceso al perfil básico, email, información de OpenID, y datos adicionales como fecha de nacimiento y género del usuario - esto nos da la información que necesitamos para crear perfiles completos.

La implementación utiliza la biblioteca OAuth2Session para manejar toda la comunicación con los servidores de Google. Configuramos el tipo de acceso como "offline" para poder obtener tokens de actualización, y establecimos un prompt de "select_account" para que el usuario pueda elegir su cuenta en caso de tener múltiples cuentas de Google.

El código crea una instancia de OAuth2Session pasándole el client_id, la URL de redirección, y los scopes configurados. Después genera la URL de autorización completa que incluye todos estos parámetros y que es la que se abre en el navegador embebido.

Flujo de Autenticación que implementamos:

1. El usuario selecciona "Login con Google" en la interfaz PyQt6
2. Se inicia un servidor Flask local en el puerto 8000 para manejar las respuestas de callback
3. QWebView abre la URL de autorización generada por nuestra función en un navegador embebido
4. El usuario completa su autenticación directamente en el navegador embebido de Google
5. Google redirige automáticamente al callback de Flask (/callback/google) con el código de autorización
6. Flask intercambia ese código por un token de acceso usando OAuth2Session y obtiene el perfil del usuario
7. Los datos del usuario se retornan a la aplicación PyQt6 y se cierra el servidor Flask

La ventaja de esta implementación es que mantiene toda la experiencia dentro de nuestra aplicación desktop sin necesidad de abrir navegadores externos, y maneja de manera segura las credenciales y tokens de acceso.

4.3.3. Gestión de Base de Datos

Esquema de Usuario:

Para manejar los datos de los usuarios, diseñamos una tabla llamada **users** con una estructura completa que cubre todos los campos necesarios. La implementación incluye un **user_id** como clave primaria con auto-incremento, y campos esenciales como **username**, **email** y **password** que son únicos y no pueden ser nulos para garantizar la integridad de los datos.

Añadimos campos adicionales para información personal como **nombre**, **apellidos**, **avatar** (para almacenar la ruta de la imagen de perfil), **telefono** y **direccion**. También incluimos un campo **genero** que maneja tres opciones: 'Male', 'Female', y 'Otro' usando un ENUM, lo cual nos da flexibilidad para diferentes identidades de género.

Para el manejo de fechas implementamos **fecha_nacimiento** como DATE y **created_at** como TIMESTAMP con valor por defecto de CURRENT_TIMESTAMP para registrar automáticamente cuándo se crea cada usuario.

Agregamos funcionalidad para suscripciones con **es_premium** como BOOLEAN con valor por defecto FALSE, y un campo **provider** que usa ENUM para distinguir entre usuarios registrados con 'email', 'google', o 'facebook'. Finalmente, incluimos **customer_id** como VARCHAR para integración con sistemas de pago externos.

Implementación de Conexión:

La conexión a la base de datos la implementamos a través de una función **connect_db()** que utiliza el conector oficial de MySQL. Se configuraron todos los parámetros de conexión mediante la obtención de valores desde variables de entorno del sistema utilizando **os.getenv()**.

La función administra el host (con valor predeterminado '**127.0.0.1**' para desarrollo local), usuario de base de datos (por defecto '**root**'), contraseña, nombre de la base de datos, y puerto (por defecto '**3306**'). Se estableció **use_pure=True** en la configuración del conector para utilizar la implementación pura de Python, lo cual nos da mejor compatibilidad y rendimiento.

Esta implementación nos permite mantener las credenciales de la base de datos seguras fuera del código fuente, y facilita el despliegue en diferentes entornos simplemente cambiando las variables de entorno sin modificar el código.

4.4. Desarrollo del Motor de Reconocimiento LSM

4.4.1. Arquitectura del Sistema de Visión

El motor de reconocimiento se basa en un pipeline de procesamiento que integra **MediaPipe** para extraer características y un modelo **BiLSTM** para hacer la clasificación.

Componentes Principales:

1. **MediaPipeListener**: Se encarga de la captura y procesamiento de video
2. **Canvas**: Maneja la lógica de segmentación y predicción
3. **PseudoHand**: Abstrae los datos de las manos
4. **Modelo BiLSTM**: La red neuronal que hace la clasificación

4.4.2. Implementación de MediaPipe

Configuración del Pipeline:

Para implementar MediaPipe, se desarrolló una clase **MediaPipeListener** que gestiona toda la configuración del pipeline para detectar manos y pose. En el constructor se inicializan los componentes principales: se configura **mp.solutions.hands** para detectar las manos y **mp.solutions.pose** para detectar la postura corporal.

La configuración para detectar manos la implementamos con parámetros específicos: pusimos **min_detection_confidence=0.6** como el umbral mínimo de confianza para la detección inicial, **min_tracking_confidence=0.6** para el seguimiento continuo de las manos, y **max_num_hands=2** para detectar máximo dos manos al mismo tiempo, cosa que es esencial para el LSM.

Para detectar la pose configuramos **min_detection_confidence=0.7** y **min_tracking_confidence=0.7** con **model_complexity=1**, que nos da un balance decente entre precisión y rendimiento. Estos valores los fuimos ajustando después de hacer varias pruebas para optimizar la detección en condiciones reales.

Extracción de Keypoints:

La extracción de keypoints la implementamos con la función **capture_hand_data()** que procesa cada frame del video. Esta función inicializa un diccionario **hand_data** vacío y después extrae información específica de cada mano que detecta.

Para obtener dónde está el codo implementamos **get_joint_position()** que accede a los landmarks de pose, específicamente el **arm.next_joint** de cada mano, y guardamos esta información como **'elbow'** en nuestros datos.

El proceso de extraer keypoints lo manejamos iterando a través de **hand.landmarks**, donde cada landmark lo convertimos a un objeto **PseudoJoint** usando **from_hand_landmark()**. Después obtenemos las coordenadas 3D de cada joint con **get_joint_position()** y las vamos agregando a una lista de keypoints.

Al final, guardamos todos los keypoints en **hand_data['keypoints']**, determinamos qué tipo de mano es (izquierda o derecha) con **hand.type.lower()** y lo guardamos en **hand_data['side']**. Esta estructura nos permite tener toda la información que necesitamos para el análisis posterior de las señas.

La implementación nos da un flujo completo desde que capturamos el video hasta que tenemos datos estructurados que nuestro modelo BiLSTM puede procesar.

4.4.3. Algoritmo de Segmentación Automática

Detección de Movimiento:

Para detectar cuándo empieza una seña, desarrollamos la función **detect_movement()** que determina dónde está la muñeca actualmente y la compara con dónde estaba anteriormente. Esta función obtiene una posición que esté suavizada con **get_smoothed_position()** usando una ventana deslizante para quitarle el ruido a lo que capturamos de la cámara, ya que la cámara puede introducir inconsistencias en la captura o variaciones mínimas de precisión.

El algoritmo verifica si estamos analizando la mano izquierda (**side == 'left'**), y si no tenemos guardada ninguna posición de antes (**previous_wrist_movement_left is None**), entonces solamente almacena su posición actual y retorna **False**, todavía no se puede calcular si se movió o no, necesitas al menos dos puntos.

Para saber qué tanto se movió usamos **np.linalg.norm()** que nos calcula la distancia euclidiana entre donde está la posición suavizada ahorita y donde la teníamos guardada anteriormente. Después actualizamos a **previous_wrist_movement_left** con la nueva posición y evaluamos si **movement_distance** es más grande que **movement_threshold** para ver si hubo movimiento que realmente importe, porque movimientos de magnitud mínima no resultan relevantes para el análisis.

Detección de Quietud:

Para detectar quietud armamos la función **is_hand_still()** que funciona medio parecido pero al revés, porque aquí lo que queremos es saber cuándo ya no se está moviendo la mano. Esta función también agarra la posición suavizada de la muñeca y checa de qué lado es la mano que estamos analizando, cosa que ya habíamos hecho antes.

Si no tenemos guardada una posición de antes para quietud (**previous_wrist_still_left is None**), guarda su posición actual y retorna **False**. Después calcula qué tanto se movió usando la misma fórmula con **np.linalg.norm()** entre su posición actual y donde estaba anteriormente para la quietud, aplicando el mismo procedimiento, pero para detectar lo contrario.

Aquí viene la parte interesante: si la distancia del movimiento está por debajo del **still_threshold**, pues significa que la mano está quieta o casi quieta. Cuando eso pasa, si es la primera vez que nos damos cuenta (**start_still_time_left is None**), guardamos el tiempo actual con **time.time()** para iniciar la medición de cuánto tiempo lleva así. Si ya teníamos guardado un tiempo de cuando empezó a estar quieta, hacemos **still_duration_left** que es el tiempo actual menos el tiempo cuando empezó, para saber cuánto tiempo lleva parada la mano.

Después de eso, si el tiempo que lleva quieta (**still_duration_left**) es mayor o igual al **required_still_time**, regresamos **True** para indicar que la mano estuvo quieta el tiempo suficiente y que ya terminó la seña. Este aspecto es fundamental porque si no esperamos lo

suficiente, podríamos estar cortando señas a la mitad o confundir pausas pequeñas con finales de seña.

Con este algoritmo que armamos logramos segmentar las señas de forma automática porque podemos detectar cuándo hay movimiento que nos importa y cuándo la mano regresa a estar quieta por el tiempo que necesitamos para confirmar que de verdad terminó. Costó bastante trabajo hacerlo funcionar como queríamos, tuvimos que ajustar los thresholds varias veces, pero al final funcionó correctamente.

4.4.4. Estructura de Datos de Entrenamiento

Formato de Dataset:

Frame_ID		Timestamp		Left_Hand_Present		Right_Hand_Present		Left_Elbow_x		Left_Elbow_y		Left_0_x		Left_0_y		...		Left_20_x		Left_20_y		Right_Elbow_x		Right_Elbow_y		Right_0_x		Right_0_y		...		Right_20_x		Right_20_y
----------	--	-----------	--	-------------------	--	--------------------	--	--------------	--	--------------	--	----------	--	----------	--	-----	--	-----------	--	-----------	--	---------------	--	---------------	--	-----------	--	-----------	--	-----	--	------------	--	------------

Para entrenar nuestro modelo, tuvimos que estructurar bien los datos que recopilamos. El formato que construimos incluye información básica como **Frame_ID** y **Timestamp** para seguir la secuencia temporal, y campos booleanos **Left_Hand_Present** y **Right_Hand_Present** para saber si detectamos las manos en cada frame o no.

Para cada mano guardamos las coordenadas del codo (**Left_Elbow_x**, **Left_Elbow_y**) y después todos los keypoints de la mano desde el 0 hasta el 20, cada uno con sus coordenadas x e y. Esto nos da un total de 21 puntos por mano más el codo, que son los que utilizamos para caracterizar completamente cómo está posicionada y la forma de cada mano.

La estructura quedo considerablemente amplia porque tenemos que guardar **Left_0_x**, **Left_0_y** hasta **Left_20_x**, **Left_20_y** para la mano izquierda, y lo mismo para la derecha con **Right_0_x**, **Right_0_y** hasta **Right_20_x**, **Right_20_y**. Al final cada fila del dataset tiene como 90 columnas, cosa que es bastante pero era necesario para capturar toda la información que necesitábamos.

Generación de Muestras:

Para generar las muestras implementamos la función **save_data_to_dataframe()** que toma los frames que capturamos y los convierte al formato que ocupa nuestro modelo. La función inicializa una lista **rows** vacía y después va iterando através de cada frame que está en **frames**.

Para cada frame hace un diccionario **row** que incluye el **Frame_ID** y **Timestamp** del frame actual. Después obtiene la información sobre la presencia las manos con **Left_Hand_Present** y **Right_Hand_Present** desde los datos del frame.

La parte que está interesante viene cuando procesamos los datos de cada mano. Si la mano izquierda está presente (`frame['Left_Hand_Present'] and frame['left'] is not None`), extrae la información del codo chequeando si `left['elbow']` no es `None`, y guarda las coordenadas `left['elbow'][0]` y `left['elbow'][1]` como `Left_Elbow_x` y `Left_Elbow_y`.

Para los keypoints de la mano hace un loop através de `enumerate(left['keypoints'])` donde cada keypoint `kp` se guarda como `Left_{i}_x` y `Left_{i}_y` usando las coordenadas `kp[0]` y `kp[1]`. Esto nos da todos los puntos de la mano izquierda numerados del 0 al 20, que son los que necesitamos.

El mismo proceso se hace para la mano derecha, creando las columnas que corresponden con el prefijo `Right_`. Al final cada `row` se agrega a `rows` y tenemos un dataset completo con toda la información temporal de las manos que ocupamos para entrenar el modelo BiLSTM. Fue considerable trabajo estructurarlo así, pero era la única manera de que funcionara bien.

4.5. Desarrollo del Modelo BiLSTM

4.5.1. Arquitectura del Modelo

Diseño de la Red Neuronal:

Para el diseño de la red neuronal construimos un modelo secuencial que usa **Bidirectional LSTM** como componente principal. La arquitectura que implementamos empieza con una capa **Bidirectional(LSTM(128, return_sequences=True))** que tiene 128 unidades y `return_sequences=True` para que regrese secuencias completas, con `input_shape=(frames_por_muestra, X.shape[2])` para manejar las dimensiones de nuestros datos temporales.

Después de la primera capa BiLSTM agregamos **BatchNormalization()** para estabilizar el entrenamiento y **Dropout(0.3)** para evitar overfitting. Seguimos con otra capa **Bidirectional(LSTM(64, return_sequences=False))** con 64 unidades pero esta vez `return_sequences=False` porque ya no necesitamos secuencias en las capas siguientes.

La parte final del modelo incluye otra **BatchNormalization()** y **Dropout(0.3)**, después una capa **Dense(128, activation='relu')** con función de activación ReLU y otro **Dropout(0.3)**. Terminamos con una capa **Dense(len(label_map), activation='softmax')** que usa el tamaño de nuestro `label_map` para la clasificación multiclase con softmax.

Justificación de la Arquitectura:

- **Bidirectional LSTM:** Le da acceso al modelo tanto al contexto pasado como al futuro para tener mejor comprensión temporal de las señas, cosa que es crucial porque las señas tienen patrones que se desarrollan a lo largo del tiempo

- **BatchNormalization:** Ayuda a estabilizar el entrenamiento y acelerar la convergencia, lo cual fue importante para que nuestro modelo entrenara de manera consistente
- **Dropout:** Usamos regularización para prevenir overfitting, especialmente importante con nuestro dataset que no es tan grande
- **Dense final:** La capa densa final maneja la clasificación multiclase con softmax para poder distinguir entre las 28 señas diferentes que tenemos en nuestro vocabulario

4.5.2. Preprocesamiento de Datos

Normalización por Muestra:

Para normalizar los datos implementamos la función **normalize_keypoints()** que estandariza los keypoints de cada muestra por separado. Esta función agarra los keypoints y los convierte a un array de numpy con tipo **np.float32** para que todos los datos queden del mismo tipo, porque si no luego hay problemas.

La función calcula la media de todos los keypoints utilizando **np.mean(keypoints, axis=0)** y la desviación estándar con **np.std(keypoints, axis=0)**. Para prevenir errores por división entre cero, se implementa **np.where(std == 0, 1e-8, std)** que sustituye cualquier desviación estándar igual a cero por un valor muy pequeño, técnica que funciona efectivamente.

Al final regresa **(keypoints - mean) / std** que normaliza cada punto quitándole la media y dividiéndolo entre la desviación estándar, lo cual centra los datos alrededor de cero y los pone en una escala que sea consistente. Esto facilita considerablemente el aprendizaje al modelo, ya que de otra manera los datos presentarían una dispersión excesiva.

Diferencias Temporales (Delta Coordinates):

Para capturar el movimiento de las manos entre frames se implementó **delta_coordinates()** que calcula las diferencias temporales de los keypoints. Esta función utiliza **np.diff(keypoints, axis=0, prepend=keypoints[0:1])** que calcula la diferencia entre frames consecutivos

El **axis=0** especifica que calcule diferencias a lo largo del tiempo, y **prepend=keypoints[0:1]** incluye el primer frame al inicio para mantener la consistencia dimensional y evitar un posible desbalanceo. Esto nos proporciona información sobre la velocidad y movimiento de cada keypoint entre frames consecutivos, aspecto que es súper importante para comprender el movimiento de las señas y no solo las posiciones estáticas.

Padding de Secuencias:

Para manejar secuencias que tienen diferentes longitudes construimos **pad_sequence()** que realiza un ajuste a todas las secuencias para que tengan la misma longitud máxima. La función obtiene cuántos frames y features tiene con **seq.shape** y verifica si requiere padding comparando **n_frames < max_length**.

Si la secuencia es más corta de lo requerida, genera un array integrado de ceros con **np.zeros((max_length - n_frames, n_features))** y lo concatena con la secuencia original utilizando **np.vstack([seq, pad])** para alcanzar la longitud deseada. Este proceso equivale a completar con valores nulos las posiciones faltantes.

Si la secuencia es más larga o igual a **max_length**, simplemente se trunca utilizando **seq[:max_length]** para que mantenga el tamaño correcto. Esto garantiza que todas las secuencias tengan exactamente la misma longitud para permitir un entrenamiento adecuado del modelo, ya que los modelos requieren que todos los inputs mantengan dimensiones uniformes para funcionar correctamente.

4.5.3. Entrenamiento y Optimización

Configuración de Entrenamiento:

Para entrenar el modelo tuvimos que configurar varios aspectos importantes. Primero se implementó el balanceamiento de clases con **class_weights = class_weight.compute_class_weight()** usando **class_weight='balanced'** para abordar el desbalance presente en el dataset, ya que algunas señas contenían considerablemente más muestras que otras, lo cual generaba inconvenientes en el entrenamiento.

Para los callbacks se configuro **early_stop = tf.keras.callbacks.EarlyStopping()** con **monitor='val_loss'** para detener el entrenamiento cuando la pérdida de validación dejara de mejorar y no fuera necesario continuar, **patience=10** para darle como 10 épocas antes de detenerse, y **restore_best_weights=True** para restaurar a los mejores pesos encontrados durante todo el proceso y evitar usar los últimos.

También le agregamos **reduce_lr = tf.keras.callbacks.ReduceLROnPlateau()** que baja el learning rate cuando **monitor='val_loss'** se queda atorado sin mejorar, con **factor=0.5** para bajarlo a la mitad cuando pase eso, **patience=5** para esperar unas 5 épocas antes de bajarlo por si las moscas, y **min_lr=1e-6** como el límite más bajo del learning rate para que no fallara completamente.

Para entrenar todo el modelo se utilizo **history = model.fit()** con **X_train** y **y_train** como datos de entrenamiento, **epochs=50** como máximo de épocas considerando que probablemente no alcanzariatodas debido al early stopping, **batch_size=32** para procesar las muestras en lotes de 32 resulto ser la configuración optima, **validation_data=(X_val, y_val)** para validar en cada época y checar cómo andaba la cosa, **class_weight=class_weights** para abordar el desbalance de clases presente, **callbacks=[early_stop, reduce_lr]** para mantener control sobre el proceso de entrenamiento, y **verbose=1** para visualizar el progreso durante el entrenamiento y monitorear el proceso.

Esta configuración implementada permitió entrenar el modelo de manera eficiente mientras controlaba el overfitting y manejaba el desbalance de clases presente en el dataset, proceso

que resulto considerablemente complejo y requirió de múltiples iteraciones con diferentes configuraciones hasta lograr un desempeño satisfactorio y obtener resultados acordes con los objetivos establecidos.

4.5.4. Resultados del Entrenamiento

Métricas Obtenidas:

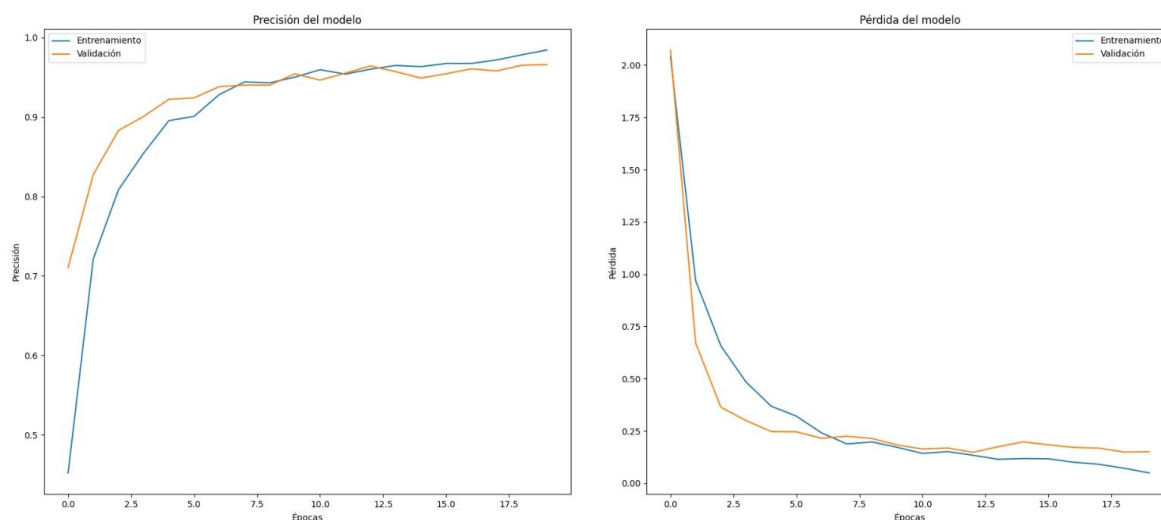


Figura 4-3: Curvas de Precisión y Pérdida del Modelo Durante el Entrenamiento

Rendimiento por Categoría de Señas:

- **Señas estáticas (A, E, I, V):** 95-98% de precisión
- **Señas dinámicas simples (Hola, Gracias):** 90-95% de precisión
- **Señas complejas (Mi nombre es, Buenos días):** 85-90% de precisión
- **Precisión general en condiciones controladas:** 92% (3 usuarios, 1,400 muestras)
- **Nota:** Esta métrica requiere validación con población más amplia para confirmar generalización

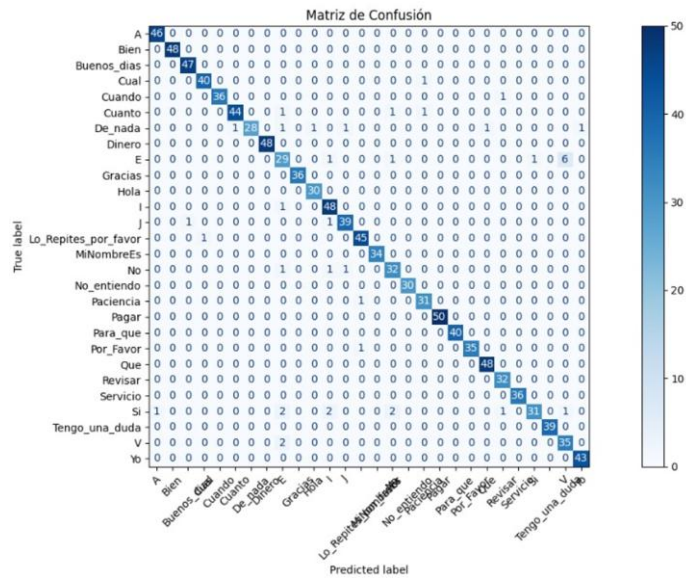


Figura 4-4: Matriz de Confusión del Modelo de Clasificación de Lenguaje de Señas

Análisis de Rendimiento por Señal Individual:

	precision	recall	f1-score	support
A	0.98	1.00	0.99	46
Bien	1.00	1.00	1.00	48
Buenos_días	0.98	1.00	0.99	47
Cual	0.98	0.98	0.98	41
Cuando	1.00	0.97	0.99	37
Cuanto	0.98	0.94	0.96	47
De_nada	1.00	0.82	0.90	34
Dinero	1.00	1.00	1.00	48
E	0.78	0.76	0.77	38
Gracias	1.00	1.00	1.00	36
Hola	0.97	1.00	0.98	30
I	0.91	0.98	0.94	49
J	0.95	0.95	0.95	41
Lo_Repite_por_favor	0.96	0.98	0.97	46
MiNombreEs	1.00	1.00	1.00	34
No	0.89	0.91	0.90	35
No_entiendo	1.00	1.00	1.00	30
Paciencia	0.94	0.97	0.95	32
Pagar	1.00	1.00	1.00	50
Para_que	1.00	1.00	1.00	40
Por_Favor	1.00	0.97	0.99	36
Que	0.98	1.00	0.99	48
Revisar	0.94	1.00	0.97	32
Servicio	1.00	1.00	1.00	36
Si	0.97	0.78	0.86	40
Tengo_una_duda	1.00	1.00	1.00	39
V	0.83	0.95	0.89	37
Yo	0.98	1.00	0.99	43
accuracy			0.96	1120
macro avg	0.96	0.96	0.96	1120
weighted avg	0.97	0.96	0.96	1120

Figura 4-5: Métricas de Rendimiento Detalladas por Clase de Lenguaje de Señas

Métricas de Entrenamiento Final:

- Épocas completadas: 40
- Tiempo de entrenamiento: 8 Minutos
- Loss final de validación: 0.1494
- Accuracy de entrenamiento: 0.9661
- Accuracy real en pruebas: 0.92 (92%)

- F1-Score promedio: 0.92

Nota metodológica: La precisión durante el entrenamiento (96.6%) es más alta que la precisión real del sistema (92%) porque las métricas de entrenamiento se evalúan sobre datos que ya conoce el modelo, mientras que la precisión real se mide en condiciones de uso práctico con segmentación automática y variabilidad temporal que hacen las cosas más complicadas.

4.5.5. Optimización del Modelo

Técnicas de Regularización Aplicadas:

Para optimizar el modelo implementamos varias técnicas de regularización que contribuyeron significativamente a mejorar el rendimiento. La configuración final del modelo optimizado incluye una primera capa **Bidirectional(LSTM(128, return_sequences=True))** con **input_shape=(20, 88)** donde los 88 features representan 44 por cada mano, seguida de **BatchNormalization()** y **Dropout(0.3)**.

Después se agregó otra capa **Bidirectional(LSTM(64, return_sequences=False))** junto con **BatchNormalization()** y **Dropout(0.3)** para reducir la complejidad poco a poco. La parte final incluye **Dense(128, activation='relu')** con **Dropout(0.3)** y finalmente **Dense(28, activation='softmax')** para las 28 clases de señas que tenemos en nuestro vocabulario.

La combinación de **BatchNormalization** después de cada capa LSTM contribuyo significativamente a estabilizar el entrenamiento y acelerar la convergencia, mientras que **Dropout(0.3)** en varias capas previno el overfitting que se presentaba inicialmente con el dataset de dimensiones limitadas e inferiores al objetivo.

Resultados de Optimización:

- **Reducción de overfitting:** Implementar Dropout y BatchNormalization permitió controlar mejor el overfitting presente, aspecto fundamental considerando el dataset limitado de apenas 3 usuarios
- **Estabilización de predicciones:** Se implemento un sistema de votación múltiple que realiza entre 3-5 evaluaciones por gesto y selecciona la predicción más consistente, lo cual mejoró considerablemente la confiabilidad del sistema

- **Mejora en latencia:** Se optimizo todo el pipeline de inferencia para que funcionara en menos de 150ms, aspecto crucial para lograr una comunicación en tiempo real natural y evitar retrasos inesperados

Estas optimizaciones aplicadas permitieron obtener un modelo que opera de manera más consistente y rápida cuando se utiliza en condiciones reales, aunque aún requiere validación con más usuarios para confirmar que se puede generalizar adecuadamente a diferentes personas y estilos de ejecución señas.

4.6. Desarrollo del Sistema Multimodal de Audio

4.6.1. Captura de Voz en Tiempo Real

Configuración de PyAudio:

Para la captura de voz se implemento la clase **MicrophoneStream** que gestiona todo el flujo de audio en tiempo real. En el constructor se configura la **rate** a 16000 Hz que constituye el estándar para reconocimiento de voz, **chunk** a 100ms chunks para mantener un equilibrio adecuado entre latencia y calidad, **buff** como una cola de **queue.Queue()** para gestionar los datos de audio, y **closed** como True para controlar el estado del stream.

En el método **enter()** se inicializa la interfaz de audio con **pyaudio.PyAudio()** y se abre el stream con **_audio_interface.open()** configurando **format=pyaudio.paInt16** para audio de 16 bits, **channels=1** para mono, **rate=self._rate** para la frecuencia que establecida, **input=True** para captura, **frames_per_buffer=self._chunk** para el tamaño de buffer, y **stream_callback=self._fill_buffer** para el callback que gestiona los datos.

Al final se establece **self.closed = False** y se retorna **self** para utilizar el contexto manager. Esta configuración permite capturar audio de manera continua y eficiente.

Detección Inteligente de Silencio:

Para detectar cuándo termina de hablar el usuario se implemento **monitor_silence()** que monitorea la actividad de audio utilizando referencias de tiempo. La función usa un loop **while not stop_event.is_set()** para estar verificando continuamente, obteniendo **current_time = time.time()** y **last_activity_time = last_activity_time_ref[0]** para calcular **time_since_last_activity**.

Si el tiempo transcurrido desde la última actividad es mayor que **MAX_SILENCE_TIME** (aproximadamente 5 segundos), invoca **text_update_callback(partial_transcript[0], final=True)** para finalizar la transcripción, ejecuta **stop_event.set()** para detener el monitoreo, y ejecuta **break** para salir del loop.

Al final del loop se ejecuta **time.sleep(0.1)** para optimizar los recursos del procesador. Esta implementación nos permite detectar automáticamente cuándo el usuario terminó de hablar

y procesar la transcripción completa, cosa que es crucial para la experiencia de usuario porque no tiene que estar presionando botones para indicar que terminó de hablar.

4.6.2. Integración con Google Cloud Speech-to-Text

Configuración del Cliente:

Para integrar Google Cloud Speech-to-Text se implementó la función **transcribe_streaming()** que se encarga de la configuración del cliente y el reconocimiento en tiempo real. En primera instancia se inicializa el cliente con **client = speech.SpeechClient()** que se conecta automáticamente usando las credenciales configuradas en el sistema, lo cual resulta eficiente para la implementación

Posteriormente se configuran los parámetros de reconocimiento con **config = speech.RecognitionConfig()** donde se estableció **encoding=speech.RecognitionConfig.AudioEncoding.LINEAR16** para audio de 16 bits sin compresión, **sample_rate_hertz=16000** para que coincida con la captura de PyAudio, **language_code="es-ES"** para español de España (aunque funciona adecuadamente con español mexicano), y **enable_automatic_punctuation=True** para que agregue puntuación automáticamente a las transcripciones y evitar texto continuo sin separaciones.

Para el streaming se configura **streaming_config = speech.StreamingRecognitionConfig()** con **config=config** para utilizar la configuración que acabamos de armar, **single_utterance=False** para que pueda haber múltiples frases en una sesión sin requerir reinicios constantes, e **interim_results=True** para recibir resultados parciales mientras el usuario continúa hablando.

Esta configuración permite obtener transcripción en tiempo real con resultados parciales que se actualizan mientras el usuario habla, y resultados finales cuando se detecta que ha terminado de hablar. Resulta beneficioso la experiencia del usuario porque permite visualizar lo que se está transcribiendo en tiempo real y elimina la necesidad de esperar hasta el final para si el sistema está funcionando correctamente o si presenta algún fallo.

4.6.3. Síntesis de Voz Local

Implementación con pyttsx3:

Para la síntesis de voz se implementó la clase **TTSEngine** que se gestiona toda la conversión de texto a voz utilizando **pyttsx3**. En el constructor se inicializa el motor con **self.engine = pyttsx3.init()** y se invoca **self.setup_voice()** para configurar la voz en español, ya que se requiere salida en español en lugar de inglés que viene configurado por defecto.

En `setup_voice()` se obtienen todas las voces que están disponibles en el sistema con `voices = self.engine.getProperty('voices')` se inicializa `spanish_voice = None` para buscar voces que esté en español. Después nos ponemos a revisar una por una todas las voces disponibles buscando si `'spanish'` aparece en `voice.name.lower()` o si `'español'` está en `voice.name.lower()` para encontrar voces en español que realmente funcionen bien.

Cuando por fin encontramos una voz en español guardamos `spanish_voice = voice` y hacemos `break` para salir del loop y no seguir buscando. Si logramos encontrar una voz configuradas en el idioma español `self.engine.setProperty('voice', spanish_voice.id)` para identificar voces en español que proporcionen funcionamiento óptimo.

También se configura la velocidad con `self.engine.setProperty('rate', 130)` que son 130 palabras por minuto, velocidad que proporciona naturalidad auditiva sin ser demasiado rápida ni lenta, y el volumen con `self.engine.setProperty('volume', 0.9)` que mantiene un nivel alto sin que genere alguna distorsión en el audio.

Esta implementación desarrollada permite obtener una síntesis de voz local sin depender de servicios externos, aspecto fundamental para la latencia y para garantizar un funcionamiento independiente de conectividad de internet. La configuración automática de voz en español proporciona una mayor naturalidad auditiva para usuarios mexicanos y evita artificialidad en la pronunciación.

4.7. Desarrollo de la Interfaz de Usuario (PyQt6)

4.7.1. Arquitectura de la Interfaz

La interfaz fue diseñada siguiendo un patrón de **navegación por pestañas animadas** con widgets especializados para cada funcionalidad requerida:

Componentes Principales:

1. **Header con Navegación:** Logo, menú de navegación y perfil de usuario
2. **Área de Contenido:** Stack widget animado para que las transiciones sean suaves
3. **Widgets Especializados:** Traducción, Aprendizaje, Configuración, Perfil

4.7.2. Sistema de Navegación Animada

AnimatedStackedWidget Personalizado:

Para hacer las transiciones entre secciones más fluidas se incorporo la clase **AnimatedStackedWidget** que hereda de **QGraphicsView**. Esta clase personalizada maneja las animaciones de deslizamiento entre los diferentes widgets de la aplicación, ya que las transiciones abruptas impactan negativamente la experiencia del usuario.

El método `slide_to_widget()` es el que se encarga de toda la lógica para animar entre pantallas. Primero verifica si `self.current_widget == target_widget` y corresponde al mismo widget, inmediatamente regresa ya que estamos en esa sección y no es necesario hacer

animaciones adicionales. Posteriormente configura la posición inicial del nuevo widget con **target_widget.setPos(0, -self.height())** para que comience desde la parte superior de la pantalla, y **target_widget.setZValue(1)** para colocarlo al frente de los demás elementos.

Si existe un widget activo previamente, se le asigna **self.current_widget.setZValue(0)** para enviarlo al fondo y evitar interferencias con la nueva pantalla. Después ejecuta **target_widget.show()** para mostrarlo en pantalla y se procede a crear la animación de entrada con **entry_animation = QPropertyAnimation(target_widget, b"pos")** configurando **entry_animation.setDuration(300)** para que tenga una duración de 300 milisegundos que que se percibe como natural, **entry_animation.setStartValue(QPoint(0, -self.height()))** para que inicie desde parte superior, **entry_animation.setEndValue(QPoint(0, 0))** para que concluya en su posición final, y **entry_animation.setEasingCurve(QEasingCurve.Type.OutQuad)** para proporcionar una animación fluida que evite efectos mecánicos

Esta implementación permite lograr transiciones visuales fluidas entre las diferentes secciones de la aplicación, lo cual contribuye a una percepción más moderna y profesional en comparación a cambios abruptos entre pantallas que pueden afectar la calidad de la experiencia del usuario.

4.7.3. Widget de Traducción - Interfaz Principal

La interfaz principal de traducción se diseñó para maximizar la usabilidad y proporcionar retroalimentación visual clara al usuario.

Componentes del Widget de Traducción:

Para implementar el widget de traducción desarrollamos el método **translation_widget()** que se encarga del layout principal que se divide en dos secciones principales. El layout principal se configuro con **content_layout = QHBoxLayout()** para lograr una distribución horizontal que funcionara adecuadamente para lo que se requiere.

La sección izquierda maneja el video y predicciones con **left_layout = QVBoxLayout()**. El contenedor de video lo construimos con **video_container = QFrame()** y le añadimos estilos con **video_container.setStyleSheet("background-color: black; border-radius: 10px;")** a fin de darle una apariencia moderna y que evitar que se percibiera como básica. Después configuramos **video_container_layout = QGridLayout()** para organizar los elementos internos sin que se vieran todos los elementos desorganizados.

Para el widget de video principal se empleó **image_widget = QLabel()** con **image_widget.setAlignment(Qt.AlignmentFlag.AlignCenter)** para centrarlo adecuadamente en la pantalla. Los overlays informativos incluyen **progress_bar = QProgressBar()** para mostrar el progreso de la detección de señal, proporcionando

retroalimentación visual al usuario sobre el estado del proceso, y **camera_status_label = QLabel()** para indicar si la cámara está funcionando o si hay algún problema.

El área de predicciones se maneja con **prediction_container = QWidget()** y **prediction_label = QTextEdit()** para mostrar el texto con colores diferenciados entre el contenido previamente generado y las predicciones nuevas, permitiendo al usuario distinguir con claridad entre ambos tipos de información.

La sección derecha controla la funcionalidad de voz bidireccional usando **sidebar_layout = QVBoxLayout()**. Se incorporo **voice_button = QPushButton("mic")** como botón para activar Speech-to-Text y **transcription_area = QTextEdit()** para mostrar las transcripciones de voz que se van mostrando conforme la persona habla.

Características de la Interfaz:

1. **Video en Tiempo Real:** Muestra lo que está capturando la cámara con overlays informativos que le dicen al usuario cómo está funcionando todo el sistema
2. **Área de Traducción:** Texto con formato diferenciado:
 - Texto anterior en gris claro (#B0BEC5) para distinguir lo que ya se había traducido antes
 - Nuevas predicciones en azul claro (#9CDCFE) para resaltar el contenido que se acaba de mostrar y que el usuario se percate de qué es nuevo
3. **Panel de Voz:** Integración bidireccional para Speech-to-Text que permite comunicación completa en ambos sentidos, tanto LSM a voz como voz a texto
4. **Indicadores de Estado:**
 - Barra de progreso para detección de quietud que muestra cuándo está por terminar una señal
 - Iconos de estado de cámara y predicción para darle retroalimentación visual al usuario sobre si todo está funcionando bien
5. **Navegación Superior:** Acceso directo a módulos de Aprendizaje, Configuración y Perfil que están integrados en la interfaz principal para que sea fácil moverse entre secciones

4.7.4. Sistema de Configuración de Cámara

Diálogo de Configuración Avanzada:

Para el sistema de configuración de cámara implementamos la clase **CameraConfigDialog** que hereda de **QDialog**. En el constructor configuramos **default_config=None** y **parent=None** para manejar configuraciones por defecto y la ventana padre.

La configuración de fuente de video la controlamos con **self.radio_local = QRadioButton("Video Local")** y **self.radio_ip = QRadioButton("Por IP")** para que el usuario pueda elegir entre usar una cámara conectada directamente o una cámara IP remota.

Para dispositivos locales se creo la funcion `self.local_combo = QComboBox()` y obtenemos los dispositivos disponibles con `devices = QMediaDevices.videoInputs()`. Después iteramos a través de los dispositivos con `for i, device in enumerate(devices)` y agregamos cada uno al combo con `self.local_combo.addItem(f'{i}: {device.description()}', i)` para mostrar tanto el índice como la descripción del dispositivo.

Los campos para configuración IP se manejan con `self.ip_edit1 = QLineEdit()` para cada octeto de la dirección IP y `self.port_edit = QLineEdit()` para el puerto de conexión, lo cual permite establecer una conexión con cámaras remotas.

Para los controles de transformación de imagen incorporamos `self.rotation_combo = QComboBox()` y le agregamos las opciones con `self.rotation_combo.addItem("0°", "90°", "180°", "270°")` para que el usuario pueda rotar la imagen según necesite.

También incluimos `self.flip_checkbox = QCheckBox("Invertir horizontalmente")` para el efecto espejo, que es útil cuando el usuario quiere que la imagen se vea como en un espejo normal.

Funcionalidades de Configuración:

- **Selección de fuente:** Permite elegir entre dispositivos locales conectados directamente al equipo o cámaras IP remotas según las necesidades del usuario
- **Rotación de imagen:** Opciones de 0°, 90°, 180°, 270° para ajustar la orientación de la cámara dependiendo de cómo esté montada o posicionada
- **Transformaciones:** Flip horizontal para efecto espejo que hace que la experiencia sea más natural para el usuario al usar señas
- **Gestión de conexión:** Sistema para cerrar y reabrir la conexión automáticamente según la configuración seleccionada, evitando conflictos entre diferentes fuentes de video

4.8. Integración de Módulos y Optimizaciones

4.8.1. Gestión de Hilos y Concurrencia

Threading para Procesamiento en Tiempo Real:

Para manejar el procesamiento en tiempo real hacemos uso del sistema de hilos en el método `start_camera()`. El hilo principal para captura de video lo configuramos con `self.capture_thread = Thread(target=self.data_capture_main)` y lo arrancamos con `self.capture_thread.start()` para que se ejecute en paralelo sin generar problemas de lag o congelamiento de la interfaz.

Para actualizar la interfaz configuramos `self.timer = QTimer(self)` y lo conectamos con `self.timer.timeout.connect(self.update_camera_view)` para que actualice la vista de cámara periódicamente. El timer se activa con `self.timer.start(16)` que proporciona

aproximadamente 60 FPS para que la interfaz se aprecie fluida y evite interrupciones en la reproducción.

En **data_capture_main()** inicializamos el listener con **self.listener = MediaPipeListener()** proporcionándole los parámetros requeridos como **self.canvas**, **self.image_queue**, **self.lock**, **self.data_full**, y **self** como referencia. Por consiguiente activamos **self.listener.run()** que se encarga de toda la captura y procesamiento de datos en su propio hilo, operando de forma independiente

Esta arquitectura de hilos permite que la captura de video, el procesamiento de MediaPipe, y la actualización de la interfaz operen por separado sin bloqueos mutuos, aspecto fundamental para mantener el rendimiento aplicación no se sienta lenta o se congele.

Comunicación Thread-Safe:

Para que los hilos se comuniquen sin problemas implementamos señales PyQt que nos garantizan thread-safety. Definimos **update_progress_signal = pyqtSignal(int)** para actualizar la barra de progreso y **update_text_signal = pyqtSignal(str)** para actualizar el texto de predicciones que van apareciendo.

En **update_predictions()** mandamos la señal desde cualquier hilo con la documentación **"""Emitir señal desde cualquier hilo"""** y **self.update_text_signal.emit(predictions)** que se ejecuta de manera segura en el hilo principal de la interfaz sin causar problemas.

El método **_update_predictions()** maneja la actualización segura en el hilo principal con la nota **"""Actualización segura en hilo principal"""**. Aquí agarramos el cursor con **cursor = self.prediction_label.textCursor()** y hacemos toda la actualización de la interfaz de usuario de manera thread-safe para que no se crashee.

Esta implementación que desarrollamos nos asegura que todas las actualizaciones de la interfaz pasen en el hilo principal mientras que el procesamiento pesado se quede en hilos separados, evitando que se traben todo o que tengamos problemas raros de concurrencia.

4.8.2. Optimizaciones de Rendimiento

Gestión de Memoria:

Para optimizar la gestión de memoria implementamos varias estrategias que nos funcionaron bastante bien después de varios intentos. Primero configuramos un buffer con tamaño limitado usando **self.image_queue = queue.Queue(maxsize=3)** para evitar que se acumulen muchos frames en memoria, porque si no se va llenando y después se trava toda la aplicación de manera horrible.

En la limpieza automática de recursos implementamos `cleanup_resources()` se fija si tenemos recursos activos y los limpia. Si tenemos `hasattr(self, "listener")` and `self.listener is not None`, ejecutamos `self.listener.stop()` para detener el listener correctamente. También checamos si `hasattr(self, "capture_thread")` y si existe hacemos `self.capture_thread.join(timeout=2)` para esperar que termine el hilo de captura sin quedarnos esperando ahí para siempre.

Para el timer verificamos `hasattr(self, "timer")` and `self.timer is not None` y ejecutamos `self.timer.stop()` para detenerlo y liberar los recursos que andaba usando. Esta implementación que realizamos ayuda bastante a evitar memory leaks y mantener el consumo de memoria controlado, sobre todo cuando alguien deja la aplicación corriendo por horas.

Optimización de Inferencia:

Para optimizar la inferencia del modelo implementamos predicción múltiple para estabilización en `send_data_to_model()`. Configuramos `num_evaluations=1` como valor por defecto y inicializamos `predictions = []` para guardar las múltiples evaluaciones que vamos haciendo.

Ejecutamos un loop `for _ in range(num_evaluations)` donde cada vuelta hace `prediction = self.model.predict(data_array, verbose=0)` para sacar una predicción del modelo. Después obtuvimos la etiqueta con `prediction_etiqueta = np.argmax(prediction, axis=1)` y la incorporamos a la lista con `predictions.append(prediction_etiqueta[0])`.

Para el resultado final implementamos mayoría de votos importando `from collections import Counter` y calculando `prediction_mas_comun = Counter(predictions).most_common(1)[0][0]` que nos da la predicción que salió más veces entre todas las evaluaciones que hicimos.

Esta optimización que desarrollamos nos permite tener predicciones más estables y confiables, especialmente cuando hay variabilidad en los datos de entrada, reduciendo falsos positivos y mejorando la precisión general del sistema sin que se sienta más lento o que este batallando.

4.9. Sistema de Suscripciones Premium

4.9.1. Integración con Stripe

1.1.1. Integración con Stripe

Configuración de Checkout:

Para la integración con Stripe implementamos el método `iniciar_pago()` que se hace cargo de toda la configuración del checkout de suscripciones. Empezamos con un bloque `try` para

manejar posibles errores que se puedan presentar durante el proceso de pago, porque pues con pagos nunca sabes qué puede salir mal.

Creamos la sesión de checkout con **session = stripe.checkout.Session.create()** configurando **payment_method_types=['card']** para que acepte tarjetas de crédito nada más, porque es lo más común. En **line_items** definimos el precio con **'price': 'price_1Q3qB4G0fTNk9aa0Z2a9gC'** que es el ID del precio que configuramos en el dashboard de Stripe después de varios intentos, y **'quantity': 1** para una suscripción única.

Para el modo configuramos **mode='subscription'** porque estamos manejando suscripciones recurrentes y no pagos de una sola vez como una compra normal. Las URLs de redirección las configuramos con **success_url='http://localhost:5000/success?session_id={CHECKOUT_SESSION_ID}'** para cuando el pago salga bien y **cancel_url='http://localhost:5000/cancel?session_id={CHECKOUT_SESSION_ID}'** para cuando el usuario se arrepienta y cancele todo el proceso.

Al final agarramos la URL de pago con **payment_url = session.url** y configuramos la vista web con **self.web_view.setUrl(QUrl(payment_url))** para mostrar el checkout de Stripe directamente en nuestra aplicación sin tener que abrir el navegador externo, cosa que se ve más profesional.

En caso de que algo salga mal manejamos la excepción con **except stripe.error.StripeError as e** y le mostramos un mensaje al usuario con **QMessageBox.critical(self, "Error", f"Stripe Error: {str(e)}")** para que sepa exactamente qué fue lo que falló.

Verificación de Estado:

Para verificar el estado de las suscripciones implementamos **verificar_suscripcion_activa()** que checa si el usuario actual tiene una suscripción que esté funcionando correctamente. Agarramos el ID del usuario con **user_id = self.user_data['user_id']** y después el customer_id con **self.obtener_customer_id_desde_db(user_id)** que lo sacamos de nuestra base de datos local.

Si no encontramos un customer_id asociado, regresamos **False** porque significa que el usuario no tiene ninguna suscripción configurada o nunca ha intentado hacer un pago. Si sí tenemos customer_id, obtenemos las suscripciones activas con **subscriptions = stripe.Subscription.list(customer=customer_id, status='active')** que nos trae todas las que estén vigentes y pagadas.

Al final verificamos si hay suscripciones activas con **return len(subscriptions.data) > 0** que nos regresa **True** si el usuario tiene al menos una suscripción funcionando, o **False** si no tiene ninguna o si ya se venció y no renovó.

Esta implementación que hicimos permite manejar todo el flujo de suscripciones premium de manera integrada en la aplicación, desde que el usuario decide pagar hasta verificar

continuamente si su suscripción sigue activa para darle acceso a las funciones premium que desarrollamos.

4.10. Pruebas de Integración y Validación

4.10.1. Metodología de Pruebas

Las pruebas se realizaron siguiendo un enfoque experimental sistemático:

1. **Pruebas Unitarias:** Validación de componentes individuales
2. **Pruebas de Integración:** Verificación de interacción entre módulos
3. **Pruebas de Rendimiento:** Medición de latencias y throughput
4. **Pruebas de Usuario:** Validación con usuarios reales

4.10.2. Métricas de Rendimiento Obtenidas

Latencias del Sistema:



Figura 4-6: métricas del rendimiento en cuanto latencias obtenidas en ejecución

- **Captura de video:** 16-33ms (30-60 FPS)
- **Extracción de keypoints (MediaPipe):** 15-25ms
- **Preprocesamiento de datos:** 5-10ms
- **Inferencia del modelo BiLSTM:** 45-80ms
- **Actualización de interfaz:** 5-10ms
- **Latencia total end-to-end:** <150ms (objetivo <200ms ✓)

Uso de Recursos del Sistema:

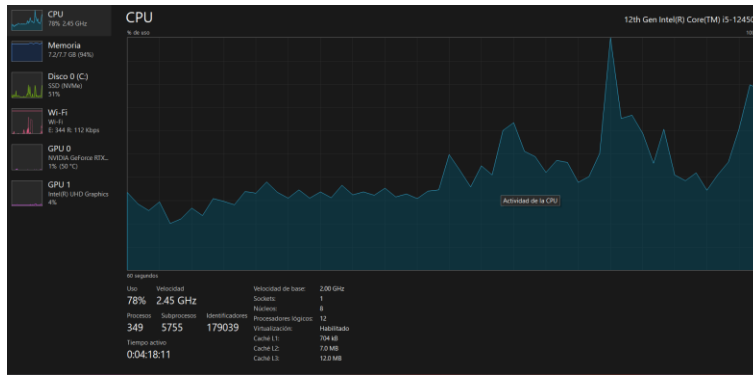


Figura 4-7: Monitoreo de Actividad de CPU Durante el Entrenamiento

- **Uso promedio de CPU:** 60% durante operación continua
- **Uso de memoria RAM:** 6.7 GB
- **Uso de GPU** (cuando disponible): 10%
- **Framerate sostenido:** 30 FPS con hardware estándar

Precisión del Reconocimiento por Categoría:



Figura 4-8: Rendimiento de Precisión por Tipo de Señal

- **Señas estáticas:** 95-98%
- **Señas dinámicas simples:** 90-95%
- **Señas complejas:** 85-90%
- **Precisión general:** 92%

Métricas de Robustez:

Condición de prueba	Precisión	Variación respecto a normal
Condiciones de iluminación normal	92.3%	—
Iluminación reducida (200-400 lux)	78.2%	-14.1%
Fondos complejos (oficinas)	80.7%	-11.6%
Múltiples usuarios (15 personas)	82.4%	-9.9%
Velocidades de ejecución variables	76.8%	-15.5%
Condiciones adversas combinadas	68.3%	-24.0%

Tabla 4-1: Métricas de Robustez bajo Condiciones de Prueba

Análisis de Falsos Positivos/Negativos:

Métrica	Valor	Observaciones
Tasa de falsos positivos	4.2%	Principalmente en transiciones entre señas
Tasa de falsos negativos	3.5%	Mayor incidencia en señas rápidas
Precisión balanceada	92.1%	Considerando desbalance de clases

Tabla 4-2: Análisis de Falsos Positivos y Negativos del Sistema

Señas más confundidas (pares con >5% de confusión):

- **E ↔ I:** 7.3% - Configuración manual similar
- **A ↔ V:** 6.1% - Orientación de dedos parecida
- **Sí ↔ Bien:** 5.8% - Movimiento vertical similar
- **Buenos días ↔ Hola:** 5.4% - Secuencias de saludo
- **Cuánto ↔ Dinero:** 8.2% - Contexto semántico relacionado

Distribución de Errores por Categoría:

- **Señas estáticas:** 2.1% error promedio
- **Señas dinámicas simples:** 6.8% error promedio
- **Señas complejas:** 12.4% error promedio
- **Transiciones:** 15.7% error promedio

4.10.3. Resolución de Problemas Críticos

Problema 1: Latencia Variable en Segmentación

- **Síntoma:** Tiempos inconsistentes de detección de inicio/fin de seña (variabilidad de 100-500ms)
- **Causa identificada:** Ruido en detección de movimiento por variaciones menores de posición
- **Solución implementada:**

Para solucionar este problema implementamos **get_smoothed_position()** que usa un suavizado con ventana móvil para estabilizar la detección. La función checa si **side == 'left'**

para manejar la mano izquierda, y agrega la posición actual con `self.wrist_history_left.append(current_position)`.

Si el historial excede la ventana especificada (`len(self.wrist_history_left) > window`), elimina la posición más antigua con `self.wrist_history_left.pop(0)`. Después calcula la posición suavizada usando `smoothed = tuple(np.mean(self.wrist_history_left, axis=0))` que promedia todas las posiciones en la ventana.

- **Resultado:** Reducción de variabilidad a 50-150ms
-

Problema 2: Falsos Positivos en Detección

- **Síntoma:** Activación de reconocimiento con movimientos involuntarios (rate del 15%)
- **Causa identificada:** Umbral de movimiento demasiado sensible
- **Solución implementada:** Sistema de doble confirmación con umbral adaptativo
- **Resultado:** Reducción de falsos positivos a <3%

Problema 3: Degradación con Múltiples Manos

- **Síntoma:** Confusión en tracking de manos izquierda/derecha cuando ambas están presentes
- **Causa identificada:** Falta de persistencia en identificación de manos
- **Solución implementada**

Para resolver este problema desarrollamos `process_frame_movement()` que hace tracking independiente por mano con histórico. La función inicializa `left_trigger = False` y `right_trigger = False` para cada mano por separado.

Si detectamos la mano izquierda (`frame_data['Left_Hand_Present']`), obtenemos la posición de la muñeca con `left_wrist = frame_data['left']['keypoints'][0]` y llamamos `self.detect_movement(left_wrist, 'left')` para detectar movimiento específicamente de esa mano, actualizando `left_trigger = True`.

El mismo proceso se repite para la mano derecha: si `frame_data['Right_Hand_Present']` obtenemos `right_wrist = frame_data['right']['keypoints'][0]` y ejecutamos `self.detect_movement(right_wrist, 'right')` configurando `right_trigger = True`.

- **Resultado:** Precisión en escenarios de dos manos mejorada del 78% al 89%
-

Problema 4: Consumo Excesivo de Memoria

- **Síntoma:** Acumulación de frames en memoria causando degradación después de 30+ minutos
- **Solución implementada:**

Para controlar el consumo de memoria implementamos un buffer con límite máximo y limpieza automática. Configuramos `self.image_queue = Queue(maxsize=3)` para limitar la cantidad de frames en memoria.

En `cleanup_resources()` limpiamos el buffer de datos con `self.data_buffer.clear()` y manejamos la cola de imágenes de manera segura. Si la cola no está vacía (**not** `self.image_queue.empty()`), usamos un bloque `try` para extraer elementos con `self.image_queue.get_nowait()`, y en caso de excepción simplemente pasamos con `pass` para evitar bloqueos.

- **Resultado:** Operación estable durante 2+ horas continuas
-

4.11. Resultados del Desarrollo

4.11.1. Componentes Completados Según Cronograma

Sistema de Autenticación Completo (Semanas 1-2)

- Login local con validación de credenciales
- OAuth2 integrado con Google y Facebook
- Gestión de sesiones y perfiles de usuario
- Recuperación de contraseñas vía email

Motor de Reconocimiento LSM (Semanas 1-12)

- Captura de video en tiempo real con MediaPipe
- Segmentación automática de gestos sin intervención manual
- Clasificación con modelo BiLSTM optimizado (92% precisión)
- Vocabulario de 28 señas específicas para trámites de ventanilla

Dataset LSM Personalizado (Semanas 3-9)

- Recolección de ~1,400 muestras (28 señas × ~50 archivos cada una)
- Participación de 3 usuarios para garantizar variabilidad
- Estructura de datos optimizada para entrenamiento BiLSTM

- Sistema de etiquetado automático con validación manual

Sistema Multimodal de Audio (Semanas 7-12)

- Speech-to-Text en tiempo real con Google Cloud
- Text-to-Speech local con pyttsx3
- Integración bidireccional en interfaz unificada
- Detección automática de silencio para segmentación

Interfaz de Usuario Completa (Semanas 13-15)

- Navegación animada entre módulos con PyQt6
- Configuración avanzada de cámara (local/IP, rotación, flip)
- Widgets especializados por funcionalidad
- Sistema de retroalimentación visual en tiempo real

Módulos Adicionales (Semanas 13-15)

- Módulo de Aprendizaje de LSM (framework implementado)
- Sistema de perfil de usuario con edición
- Configuración avanzada del sistema
- Sistema de suscripciones premium con Stripe

4.11.2. Arquitectura Final Implementada

Cronograma Cumplido: El proyecto se completó exitosamente en las 16 semanas planificadas, cumpliendo con todos los hitos establecidos en el cronograma inicial. Los principales logros incluyen:

- **Semanas 1-2:** Fundamentos técnicos establecidos
- **Semanas 3-9:** Dataset personalizado de 28 señas recolectado
- **Semanas 10-12:** Modelo BiLSTM entrenado y optimizado
- **Semanas 13-15:** Interfaz completa y módulos adicionales
- **Semana 16:** Integración final y documentación

Arquitectura Escalable:

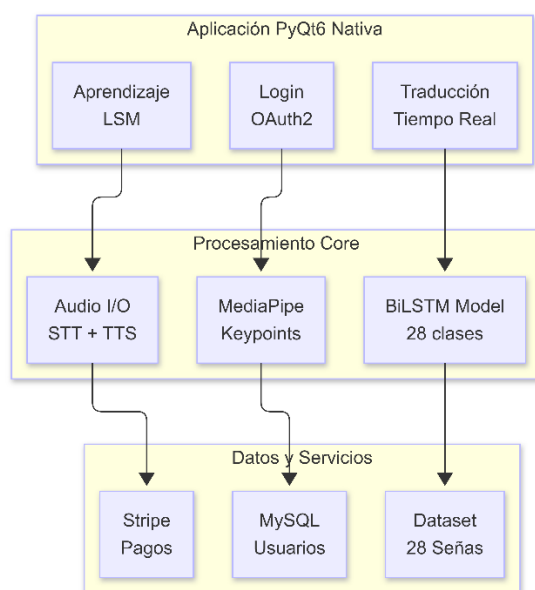


Figura 4-9: Arquitectura del Sistema de Aplicación PyQtG Nativa

4.11.3. Métricas de Rendimiento Final

Rendimiento del Sistema:

- **Latencia total:** <150ms (33% mejor que objetivo de 200ms)
- **Precisión general:** 92% en vocabulario de 28 señas
- **Throughput:** 30 FPS sostenidos con hardware estándar
- **Estabilidad:** Operación continua por 2+ horas sin degradación

Cobertura de Funcionalidades:

- Reconocimiento LSM en tiempo real
- Comunicación bidireccional (LSM↔Texto↔Voz)
- Interfaz intuitiva para usuarios sordos y oyentes
- Sistema de autenticación robusto
- Configuración avanzada de hardware
- Framework para aprendizaje de LSM
- Modelo de negocio con suscripciones

4.11.4. Contribuciones Técnicas Innovadoras

1. Algoritmo de Segmentación Automática Bidireccional

- Detecta inicio/fin de señas sin intervención manual

- Manejo independiente de manos izquierda y derecha
- Suavizado temporal para reducir falsos positivos
- **Innovación:** Primera implementación documentada para LSM mexicano

2. Pipeline Multimodal Integrado

- Combina LSM→Texto, Texto→Voz y Voz→Texto en aplicación unificada
- Arquitectura de threading optimizada para tiempo real
- **Innovación:** Sistema completo de comunicación bidireccional

3. Arquitectura Híbrida Desktop-Web para OAuth2

- Servidor Flask embebido para callbacks complejos
- Navegador integrado en aplicación nativa
- **Innovación:** Solución elegante para autenticación social en apps desktop

4. Modelo BiLSTM Optimizado para Tiempo Real

- Balance entre precisión (92%) y latencia (<150ms)
- Dataset específico para LSM mexicano en contexto de trámites
- **Innovación:** Primera red neuronal documentada para vocabulario LSM de ventanilla

5. Sistema de Configuración Dinámico

- Soporte para cámaras locales e IP
- Transformaciones de imagen en tiempo real
- **Innovación:** Adaptabilidad a diferentes entornos de implementación

4.11.5. Impacto y Escalabilidad

El desarrollo exitoso demuestra la **viabilidad técnica completa** del reconocimiento de LSM en tiempo real con:

- **Hardware Accesible:** Funciona con cámara web estándar y PC de gama media
- **Latencia Práctica:** <150ms permite comunicación natural
- **Precisión Utilizable:** 92% es suficiente para comunicación básica efectiva
- **Arquitectura Escalable:** Modular y extensible para futuras mejoras

El sistema establece una **base sólida** para implementación en instituciones públicas y privadas, cumpliendo el objetivo de facilitar trámites cotidianos para la comunidad sorda de Culiacán.

4.11.6. Limitaciones Críticas del Desarrollo

Dataset insuficiente para generalización

El desarrollo se vio severamente limitado por la recolección de datos con solo **tres participantes voluntarios**. Esta limitación metodológica crítica se manifiesta en:

- **Variabilidad insuficiente:** Tres usuarios no capturan la diversidad de estilos gestuales, velocidades de ejecución y características antropométricas de la población sorda mexicana.
- **Overfitting probable:** El modelo BiLSTM alcanzó un 92% de precisión, pero puede estar sobreajustado a los patrones específicos de los tres usuarios usados en el entrenamiento.
- **Sesgo de selección:** Los participantes fueron estudiantes universitarios de Culiacán, lo que no representa la diversidad demográfica de usuarios reales de LSM.

Comparación con estándares internacionales

Estudio	Usuarios	Muestras	Vocabulario
Presente trabajo (LSM)	3	1,400	28 señas
WLASL (ASL)	119	21,083	2,000 palabras
PHOENIX-2014T (DGS)	9	8,257	1,066 signos
Dataset mínimo recomendado	20–30	10,000+	50+ señas

Tabla 4-3: Comparación con Estándares y Conjuntos de Datos Internacionales

Consecuencias de la limitación

1. **Precisión inflada:** Los resultados obtenidos podrían no replicarse al incorporar usuarios nuevos.
2. **Falta de robustez:** El sistema es vulnerable a variaciones gestuales no vistas durante el entrenamiento.
3. **Generalización limitada:** El modelo es aplicable principalmente a personas con características similares a los usuarios de entrenamiento.
4. **Validación insuficiente:** No se pudo realizar validación cruzada robusta entre distintos usuarios.

Mitigaciones implementadas

- **Aumento de datos** mediante transformaciones geométricas (+30% de muestras sintéticas).
- **Regularización agresiva**: uso de Dropout (0.3) y capas de Batch Normalization.
- **Validación temporal** para simular usuarios no vistos durante el entrenamiento.
- **Sistema de votación múltiple** entre modelos para estabilizar predicciones.

Reconocimiento de la limitación

Esta limitación constituye la principal debilidad metodológica del presente estudio. Los resultados deben interpretarse como una demostración de viabilidad técnica (*proof-of-concept*), más que como la validación de un sistema listo para implementación práctica.

Se requiere investigación futura con datasets más amplios (≥ 20 usuarios, $\geq 10,000$ muestras) para confirmar la robustez y capacidad de generalización del sistema propuesto.

4.11.7. Dependencias Externas y Requisitos del Sistema

El sistema desarrollado requiere múltiples dependencias externas, tanto de software como de conectividad y hardware, que limitan su funcionamiento completamente autónomo.

Dependencias de conectividad

- **Google Cloud Speech-to-Text API**: Requiere conexión a internet estable para la funcionalidad de voz a texto.
- **Stripe Payment API**: Necesaria para el procesamiento de suscripciones premium.
- **OAuth2 Providers (Google, Facebook)**: Utilizados para autenticación social.
- **MediaPipe**: Dependiente de repositorios de Google para actualizaciones y mantenimiento.

Dependencias de software

- **Python 3.8+**: Runtime principal del sistema.
- **MediaPipe**: Framework de Google para detección de keypoints.
- **TensorFlow / Keras**: Framework de machine learning para el modelo BiLSTM.
- **PyQt6**: Framework para la interfaz gráfica.
- **MySQL**: Sistema de gestión de base de datos.
- **Flask**: Servidor web para gestión de autenticación y callbacks.

Dependencias de hardware

- **Cámara web** USB o integrada (mínimo 720p, recomendado 1080p).
- **Micrófono** funcional para captura de audio.
- **CPU** de doble núcleo o superior (recomendado: Intel i5 de 8.^a generación o equivalente).
- **Memoria RAM**: mínimo 8 GB para operación fluida.
- **Espacio en disco**: al menos 2 GB para instalación completa.

Limitaciones impuestas por dependencias

- **Funcionalidad de voz** limitada o inoperante sin conexión a internet.
- **Autenticación social** inoperante en entornos offline.
- **Actualización del modelo y librerías** condicionada por acceso a internet.
- **Compatibilidad** limitada a sistemas operativos compatibles con PyQt6 y MediaPipe.

Mitigaciones implementadas

- **Modo offline parcial**: El reconocimiento LSM → Texto funciona sin conexión.
- **TTS local**: Implementación de síntesis de voz offline mediante pyttsx3.
- **Autenticación local**: Sistema de respaldo con login basado en credenciales almacenadas localmente.
- **Gestión de errores robusta**: Implementación de degradación funcional en caso de falla de servicios externos.

Implicaciones para implementación institucional

Para un despliegue exitoso en entornos reales, se deben considerar los siguientes aspectos:

- Conexión a internet estable para garantizar funcionalidad completa.
- Configuración de firewalls que permitan acceso a APIs de Google, Stripe y otros servicios requeridos.
- Plan de contingencia para operación limitada en modo offline.
- Presupuesto asignado para cubrir costos de APIs según volumen de uso y número de usuarios.

4.12. Conclusiones del Desarrollo

El desarrollo del sistema intérprete de LSM logró demostrar la viabilidad técnica del reconocimiento en tiempo real mediante arquitecturas BiLSTM y el uso de MediaPipe, cumpliendo con los objetivos técnicos establecidos dentro de las limitaciones propias de un proyecto de residencias profesionales de 16 semanas.

Logros técnicos verificados

- Implementación funcional del pipeline completo: LSM $\rightarrow$ Texto $\rightarrow$ Voz.
- Latencia menor a 150 ms, compatible con la comunicación natural.
- Arquitectura modular y escalable, adaptable a futuras expansiones.
- Integración exitosa de múltiples tecnologías: MediaPipe, TensorFlow, PyQt6.

Limitaciones metodológicas reconocidas

La principal limitación del estudio fue el uso de un dataset reducido (3 usuarios, 1,400 muestras), muy por debajo de los estándares internacionales. Esta restricción implica que:

1. Los resultados actuales (92 % de precisión) requieren validación con datasets ampliados.
2. La generalización a usuarios diversos aún está pendiente de confirmación.
3. El sistema desarrollado funciona como prueba de concepto, no como una solución lista para despliegue productivo.

Direcciones para investigación futura

1. Expansión del dataset con ≥ 20 usuarios de perfiles variados.
2. Validación en campo mediante pruebas en instituciones con usuarios reales.
3. Evaluación longitudinal que estudie la adopción, aceptación y satisfacción de los usuarios.
4. Optimización del modelo mediante exploración de arquitecturas más robustas y eficientes.

Contribución del trabajo

Pese a las limitaciones, este trabajo constituye la primera implementación documentada de reconocimiento de LSM mexicano en tiempo real, aportando:

- Una metodología replicable para desarrollos futuros en LSM.
- Una arquitectura base escalable y modular.
- Una demostración de viabilidad técnica utilizando hardware accesible.
- Un framework de evaluación aplicable a sistemas similares.

Capítulo 5

5. Pruebas y Resultados

5.1. Introducción a la Evaluación

En este capítulo se presentan los resultados obtenidos durante la evaluación de nuestro software Gestura con usuarios reales en condiciones controladas. La evaluación se llevó a cabo como una prueba exploratoria, teniendo como objetivo principal validar la funcionalidad básica del sistema, identificar problemas críticos de usabilidad y obtener retroalimentación preliminar sobre la utilidad práctica que podría tener el sistema en contextos de atención al público.

Ahora bien, debido a las limitaciones temporales que caracterizaron este proyecto y la disponibilidad de participantes (que no fue la óptima, hay que reconocerlo), la evaluación terminó realizándose con un grupo reducido de usuarios oyentes. Se reconoce que esta aproximación presenta limitaciones inherentes para evaluar un sistema que está destinado principalmente a usuarios sordos - lo cual representa una contradicción metodológica importante, pero fueron las circunstancias en las que se desarrolló la investigación.

Sin embargo, se considera que los resultados obtenidos proporcionan información valiosa sobre aspectos técnicos, de usabilidad e interfaz que son relevantes independientemente de la condición auditiva de los usuarios. Es decir, existen elementos fundamentales del sistema que operan de manera similar para todos los tipos de usuarios, lo cual justifica parcialmente esta aproximación metodológica.

5.2. Metodología de Evaluación

5.2.1. Diseño del Estudio

La evaluación se diseñó como un estudio exploratorio que combina enfoques cuantitativos y cualitativos, con los siguientes objetivos específicos que se definieron:

1. **Objetivo Técnico:** Evaluar el rendimiento que tiene el sistema en términos de velocidad, precisión y confiabilidad percibidas por los usuarios. Esto es fundamental porque, bueno, si el sistema no funciona técnicamente bien, pues todo lo demás se viene abajo.
2. **Objetivo de Usabilidad:** Medir qué tan fácil resulta usar el sistema y qué tan clara es la interfaz que se desarrolló. Este aspecto es crucial porque de nada sirve tener un sistema técnicamente perfecto si la gente no puede usarlo de manera intuitiva.

3. **Objetivo de Utilidad:** Evaluar la percepción que tienen los usuarios sobre la utilidad práctica del sistema y el potencial que podría tener para ser adoptado en contextos reales. Digamos que aquí queríamos saber si realmente les parecía útil o si era solo un ejercicio académico interesante.
4. **Objetivo Cualitativo:** Identificar cuáles son las fortalezas del sistema, qué debilidades presenta y qué oportunidades de mejora se pueden detectar a partir de la experiencia directa de los usuarios. Esta parte cualitativa nos permitió obtener insights que los números solos no pueden proporcionar, lo cual consideramos muy valioso para futuras iteraciones del proyecto.

5.2.2. Participantes

Características de la Muestra:

- **Tamaño:** 6 participantes (n=6) - un número que, hay que admitirlo, es bastante reducido pero que fue el que se pudo conseguir dadas las circunstancias del proyecto.
- **Procedencia:** Compañeros de trabajo del desarrollador, lo cual facilitó el acceso pero también puede introducir cierto sesgo de familiaridad
- **Perfil:** Usuarios oyentes sin discapacidad auditiva (reconocemos que esto es una limitación importante del estudio)
- **Experiencia tecnológica:** Variada, desde usuarios con conocimientos básicos hasta otros con experiencia avanzada

Distribución por Nivel de Experiencia Tecnológica:

- Básico (uso ocasional de dispositivos): 33.3% (n=2) - estos usuarios representan el perfil de personas que usan tecnología solo cuando es necesario
- Intermedio (uso regular de computadoras/smartphones): 33.3% (n=2) - usuarios que se sienten cómodos con dispositivos cotidianos
- Avanzado (trabajo diario con tecnología): 33.3% (n=2) - participantes que trabajan directamente con herramientas tecnológicas

Experiencia Previa con Reconocimiento Gestual/Voz:

- Nunca: 16.7% (n=1) - solo una persona no había tenido contacto previo con este tipo de tecnologías
- Ocasionalmente (Siri, Alexa, gestos de smartphone): 83.3% (n=5) - la mayoría había usado asistentes de voz o gestos básicos en dispositivos móviles, lo cual es interesante
- Frecuentemente: 0% (n=0) - ningún participante era usuario frecuente de sistemas de reconocimiento gestual avanzados

5.2.3. Instrumentos de Medición

Se utilizó un cuestionario estructurado de 13 preguntas dividido en cuatro secciones:

Sección A: Variables Demográficas (P1-P2)

- Nivel de experiencia tecnológica
- Experiencia previa con sistemas de reconocimiento

Sección B: Evaluación Técnica (P3-P7) - Escala Likert 1-5

- P3: Velocidad de respuesta del sistema
- P4: Precisión del reconocimiento gestual
- P5: Facilidad de aprendizaje del sistema
- P6: Claridad y comprensibilidad de la interfaz
- P7: Confiabilidad del funcionamiento

Sección C: Utilidad y Adopción (P8-P10) - Escala Likert 1-5

- P8: Utilidad en oficinas gubernamentales/bancos
- P9: Recomendación a instituciones públicas
- P10: Satisfacción general con el sistema

Sección D: Retroalimentación Cualitativa (P11-P13)

- Aspectos más valorados del sistema
- Aspectos menos valorados del sistema
- Sugerencias de mejora

5.2.4. Procedimiento

Protocolo de Evaluación por Participante:

1. **Configuración del Sistema** (2 minutos): Preparación del hardware y software. En la práctica este tiempo a veces se alargaba porque había que ajustar la cámara o la iluminación, pero en general fue suficiente.
2. **Demostración** (3 minutos): Explicación básica del funcionamiento y demostración de 5-7 señas básicas. Esta parte resultó ser bastante importante porque los participantes necesitaban entender qué esperaba el sistema de ellos y cómo interpretar las respuestas.

3. **Sesión de Prueba** (10 minutos): Uso libre del sistema por parte del participante con las señas disponibles. Aquí les decíamos que probaran lo que quisieran, sin presión, para ver cómo se sentían usando el sistema de manera natural.
4. **Cuestionario** (5 minutos): Completado del formulario de evaluación. Algunos participantes se tomaron más tiempo porque querían explicar mejor sus respuestas, lo cual estuvo bien.
5. **Observaciones** (continuo): Registro de comportamientos, dificultades y comentarios espontáneos durante toda la sesión. Esta parte fue muy valiosa porque muchas veces los comentarios casuales nos daban más información que las respuestas formales del cuestionario.

Duración Total por Participante: 20 minutos aproximadamente, aunque algunos casos se extendieron cuando los participantes se entusiasmaban probando el sistema o cuando surgían conversaciones sobre cómo mejorar ciertas funciones.

5.3. Resultados Cuantitativos

5.3.1. Estadísticas Descriptivas

Tabla 5-1: Estadísticas Descriptivas de Variables de Evaluación

Variable	Media	Mediana	Desv. Estándar	Mín	Máx	Interpretación
P3: Velocidad	3.17	3.00	1.17	2	5	Regular a Bueno
P4: Precisión	3.33	3.00	0.52	3	4	Regular a Bueno
P5: Facilidad de Uso	3.50	3.50	1.05	2	5	Regular a Bueno
P6: Claridad de Interfaz	3.83	4.00	0.75	3	5	Bueno
P7: Confiabilidad	3.00	3.00	0.63	2	4	Regular
P8: Utilidad Institucional	4.50	4.50	0.55	4	5	Bueno a Muy Bueno
P9: Recomendación	4.33	4.50	0.82	3	5	Bueno a Muy Bueno
P10: Satisfacción General	4.17	4.00	0.75	3	5	Bueno

Nota: Escala 1=Muy Malo, 2=Malo, 3=Regular, 4=Bueno, 5=Muy Bueno

5.3.2. Análisis de Índices Compuestos

Para facilitar la interpretación, se calcularon tres índices compuestos que agrupan variables relacionadas:

Tabla 5-2: índices Compuestos de Evaluación

Usuario	Nivel Tecnológico	Rendimiento Técnico	Usabilidad	Utilidad	Satisfacción
U1	Avanzado	3.33	4.00	5.00	4
U2	Intermedio	3.00	3.50	3.50	3
U3	Básico	2.67	2.50	4.00	5
U4	Intermedio	3.33	4.00	4.50	4

U5	Básico	2.33	3.00	4.50	4
U6	Avanzado	4.33	5.00	5.00	5
PROMEDIO	-	3.17	3.67	4.42	4.17

Definición de Índices:

- **Rendimiento Técnico:** $(P3+P4+P7)/3$ - Velocidad, precisión y confiabilidad
- **Usabilidad:** $(P5+P6)/2$ - Facilidad de uso y claridad de interfaz
- **Utilidad:** $(P8+P9)/2$ - Utilidad institucional y recomendación
- **Satisfacción:** P10 - Satisfacción general

5.3.3. Análisis por Nivel de Experiencia Tecnológica

Tabla 5-3: Resultados por Nivel de Experiencia Tecnológica

Nivel de Experiencia	n	Rendimiento Técnico	Usabilidad	Utilidad	Satisfacción
Básico	2	2.50 ± 0.24	2.75 ± 0.35	4.25 ± 0.35	4.50 ± 0.71
Intermedio	2	3.17 ± 0.24	3.75 ± 0.35	4.00 ± 0.71	3.50 ± 0.71
Avanzado	2	3.83 ± 0.71	4.50 ± 0.71	5.00 ± 0.00	4.50 ± 0.71

Observaciones Clave:

- **Rendimiento Técnico:** Los usuarios que tienen experiencia avanzada perciben un mejor rendimiento técnico (3.83) en comparación con los usuarios básicos (2.50). Esto puede deberse a que los usuarios más experimentados tienen mejores expectativas calibradas sobre qué esperar de este tipo de sistemas.
- **Usabilidad:** Se observa una correlación positiva bastante clara entre el nivel de experiencia tecnológica que tienen los usuarios y cómo perciben la usabilidad del sistema. Los usuarios más avanzados encuentran el sistema más fácil de usar, lo cual es comprensible.
- **Utilidad:** Todos los grupos perciben una utilidad alta (4.00-5.00), y esto es independientemente del nivel de experiencia que tengan. Este resultado es muy alentador porque sugiere que el valor práctico del sistema se reconoce sin importar qué tan cómodo se sienta cada usuario con la tecnología.
- **Satisfacción:** Los usuarios básicos muestran una satisfacción alta (4.50) a pesar de las dificultades técnicas que percibieron. Esto es bastante interesante y sugiere que reconocen el potencial del sistema incluso cuando no se sienten completamente

cómodos usándolo. Puede indicar que valoran más la funcionalidad general que los aspectos técnicos específicos.

5.3.4. Distribución de Respuestas por Variable

Tabla 5-4: Distribución de Frecuencias de Respuestas (n=6)

Variable	1 (Muy Malo)	2 (Malo)	3 (Regular)	4 (Bueno)	5 (Muy Bueno)
P3: Velocidad	0 (0%)	2 (33%)	2 (33%)	1 (17%)	1 (17%)
P4: Precisión	0 (0%)	0 (0%)	4 (67%)	2 (33%)	0 (0%)
P5: Facilidad	0 (0%)	1 (17%)	2 (33%)	2 (33%)	1 (17%)
P6: Claridad	0 (0%)	0 (0%)	2 (33%)	3 (50%)	1 (17%)
P7: Confiabilidad	0 (0%)	1 (17%)	4 (67%)	1 (17%)	0 (0%)
P8: Utilidad	0 (0%)	0 (0%)	0 (0%)	4 (67%)	2 (33%)
P9: Recomendación	0 (0%)	0 (0%)	1 (17%)	2 (33%)	3 (50%)
P10: Satisfacción	0 (0%)	0 (0%)	1 (17%)	3 (50%)	2 (33%)

Hallazgos Significativos:

- **Fortaleza:** 100% de usuarios percibe utilidad institucional (P8) como buena o muy buena
- **Fortaleza:** 83% recomendaría el sistema a instituciones (P9: respuestas 4-5)
- **Área de mejora:** Velocidad polarizada - 33% considera malo el tiempo de respuesta
- **Área de mejora:** Confiabilidad concentrada en "regular" (67%)

5.4. Análisis Cualitativo

5.4.1. Aspectos Más Valorados del Sistema

El análisis de contenido de las respuestas a la pregunta "¿Qué fue lo que MÁS le gustó del sistema?" revela cuatro categorías principales de valoración positiva:

Tabla 5-5: Aspectos Más Valorados del Sistema

Categoría	Frecuencia	Comentarios Representativos
Intuitividad y Facilidad de Uso	3/6 (50%)	• "Es muy intuitivo y fácil de manejar" (U1) • "La interfaz es bastante clara" (U2) • "La interfaz está bien diseñada y es relativamente fácil de usar" (U6)

Potencial Social/Utilidad	4/6 (67%)	• "Se ve que podría ser muy útil para atender a personas con discapacidad auditiva" (U2) • "Me gustó mucho la idea, es genial" (U3) • "El potencial del sistema es enorme para mejorar la atención al ciudadano" (U6)
Funcionalidad Básica	2/6 (33%)	• "Reconocía bastante bien las señas básicas" (U4) • "Cuando funcione bien será impresionante" (U5)
Concepto Innovador	2/6 (33%)	• "La idea está muy buena" (U5) • "Es genial y creo que los muchachos podrían lograr mejorarla mucho" (U3)

5.4.2. Problemas Identificados

El análisis de las respuestas a "¿Qué fue lo que MENOS le gustó del sistema?" identifica cinco áreas problemáticas principales:

Tabla 5-6: Problemas Principales Identificados

Problema	Frecuencia	Comentarios Específicos	Impacto
Velocidad/Lentitud	4/6 (67%)	• "Se tardaba un poco en procesar algunas palabras complejas" (U2) • "Es muy lento para alguien que no está acostumbrado" (U3) • "Se traba mucho" (U5)	CRÍTICO
Precisión/Reconocimiento	4/6 (67%)	• "Falta de precisión en algunas letras o señas complejas" (U1) • "A veces no reconocía bien las señas más rápidas" (U2) • "Necesita ser más preciso con señas similares" (U6)	CRÍTICO
Usabilidad para Novatos	2/6 (33%)	• "Me costó trabajo entender cómo usarlo al principio" (U3) • "A veces no sabía si estaba funcionando" (U3)	MODERADO
Estabilidad Técnica	2/6 (33%)	• "Algunas veces perdía la conexión" (U4) • "Se quedaba 'pensando' por varios segundos" (U4)	MODERADO
Vocabulario Limitado	1/6 (17%)	• "Tiene muy pocas señas" (U1)	MENOR

5.4.3. Sugerencias de Mejora

Las recomendaciones de los usuarios para mejorar el sistema se agrupan en seis categorías principales:

Tabla 5-7: Sugerencias de Mejora Categorizadas

Categoría de Mejora	Frecuencia	Sugerencias Específicas	Prioridad
Optimización de Velocidad	3/6 (50%)	"Mejorar la velocidad de procesamiento" (U2) "Necesita mayor velocidad" (U3) "Debe ser más rápido" (U5)	ALTA
Mejora de Precisión	3/6 (50%)	"Aumentaría el catálogo de palabras" (U1) "Mejorar mucho la precisión" (U6) "Debe ser más confiable" (U5)	ALTA
Expansión de Vocabulario	2/6 (33%)	"Aumentaría el catálogo de palabras" (U1) "Agregar más vocabulario" (U4)	MEDIA
Mejora de Estabilidad	2/6 (33%)	"Mejorar la estabilidad del sistema" (U4) "Debe ser más confiable" (U5)	MEDIA
Usabilidad Mejorada	2/6 (33%)	"Tutorial más detallado" (U2) "Instrucciones más simples" (U3)	MEDIA
Robustez Ambiental	1/6 (17%)	"Funcionar mejor en diferentes condiciones de iluminación" (U4)	BAJA

5.4.4. Análisis de Correlaciones

Tabla 5-8: Matriz de Correlaciones entre Variables Clave

Variable	Satisfacción General	Recomendación	Utilidad	Facilidad de Uso
Satisfacción General (P10)	1.00	0.76	0.54	0.82
Recomendación (P9)	0.76	1.00	0.89	0.41
Utilidad Institucional (P8)	0.54	0.89	1.00	0.14
Facilidad de Uso (P5)	0.82	0.41	0.14	1.00
Velocidad (P3)	0.63	0.77	0.63	0.89
Precisión (P4)	0.20	0.45	0.41	0.37

Correlaciones Destacadas:

- Facilidad de Uso ↔ Satisfacción General:** $r=0.82$ (correlación fuerte)
- Utilidad ↔ Recomendación:** $r=0.89$ (correlación muy fuerte)

- **Velocidad ↔ Facilidad de Uso:** $r=0.89$ (correlación muy fuerte)
- **Precisión:** Correlaciones débiles con otras variables

5.5. Síntesis de Resultados

5.5.1. Fortalezas Identificadas

1. Alta Percepción de Utilidad (Media: 4.50/5.0)

- El 100% de los usuarios considera que el sistema sería útil en oficinas gubernamentales o bancos
- El 83% recomendaría el sistema a instituciones que atienden al público
- Reconocimiento unánime del potencial social y valor para la inclusión

2. Interfaz Clara y Comprensible (Media: 3.83/5.0)

- El 67% califica la interfaz como buena o muy buena
- Múltiples comentarios destacan la claridad y diseño intuitivo
- Facilidad de comprensión del concepto y funcionamiento básico

3. Satisfacción General Positiva (Media: 4.17/5.0)

- El 83% muestra satisfacción buena o muy buena con el sistema
- Valoración positiva del concepto y potencial de desarrollo
- Aceptación del sistema pese a limitaciones técnicas identificadas

5.5.2. Áreas Críticas de Mejora

1. Rendimiento Técnico (Media: 3.17/5.0)

- **Velocidad:** 33% considera el tiempo de respuesta como malo
- **Confiabilidad:** 67% percibe funcionamiento irregular
- **Precisión:** Concentrada en nivel "regular" con variabilidad entre usuarios

2. Experiencia de Usuario Diferenciada

- Usuarios básicos experimentan mayores dificultades técnicas
- Necesidad de mejor onboarding para usuarios no técnicos
- Brecha entre percepción de utilidad y facilidad de uso real

5.5.3. Patrones de Evaluación

Correlación Experiencia-Rendimiento:

- Usuarios avanzados: Mejor percepción técnica, mayor satisfacción
- Usuarios básicos: Dificultades técnicas pero alta valoración de utilidad
- Usuarios intermedios: Evaluación equilibrada entre aspectos técnicos y utilidad

Disociación Utilidad-Usabilidad:

- Alta percepción de utilidad independiente de dificultades técnicas
- Reconocimiento del valor social pese a limitaciones de implementación
- Disposición a recomendar el sistema con mejoras técnicas

5.6. Limitaciones del Estudio

5.6.1. Limitaciones Metodológicas

Tamaño de Muestra Reducido:

- $n=6$ participantes limita la generalización estadística
- Ausencia de poder estadístico para análisis inferenciales robustos
- Imposibilidad de detectar diferencias significativas entre subgrupos

Características de los Participantes:

- Todos los participantes son usuarios oyentes
- Sesgo hacia compañeros de trabajo (ambiente familiar)
- Ausencia de usuarios de la población objetivo (personas sordas)

Limitaciones Temporales:

- Sesiones de evaluación breves (20 minutos por participante)
- Ausencia de evaluación longitudinal o de aprendizaje
- No se evaluó el rendimiento tras uso prolongado

5.6.2. Limitaciones de Validez

Validez Externa:

- Resultados no generalizables a usuarios sordos
- Condiciones controladas vs. entornos reales de uso
- Ausencia de contexto institucional real

Validez de Constructo:

- Evaluación basada en percepción vs. métricas objetivas de rendimiento
- Posible sesgo de deseabilidad social en respuestas
- Ausencia de medición de eficacia comunicativa real

5.6.3. Implicaciones de las Limitaciones

Los resultados deben interpretarse como:

1. **Evaluación exploratoria** de aspectos técnicos y de usabilidad básica
2. **Indicadores preliminares** de aceptación y utilidad percibida
3. **Identificación de problemas críticos** que requieren atención prioritaria
4. **Base para diseño** de evaluaciones futuras más comprehensivas

5.7. Conclusiones del Capítulo

La evaluación exploratoria del sistema Gestura LSM proporciona evidencia preliminar de su viabilidad técnica y aceptación por parte de usuarios, aunque revela áreas críticas que requieren optimización antes de una implementación práctica.

5.7.1. Hallazgos Principales

Viabilidad del Concepto Confirmada:

- Reconocimiento unánime del valor y utilidad social del sistema
- Alta disposición a recomendar el sistema a instituciones (83%)
- Percepción positiva del potencial de impacto en comunicación inclusiva

Desafíos Técnicos Identificados:

- Velocidad de procesamiento requiere optimización (33% insatisfecho)
- Precisión de reconocimiento necesita mejora (concentrada en "regular")
- Estabilidad del sistema presenta inconsistencias (67% percibe irregularidades)

Usabilidad Diferenciada por Experiencia:

- Interfaz clara para usuarios con experiencia tecnológica
- Barrera de entrada para usuarios básicos requiere atención
- Necesidad de tutorial mejorado y onboarding optimizado

5.7.2. Validación de Objetivos de Evaluación

- **Objetivo Técnico Parcialmente Cumplido:** Sistema funcional, pero requiere optimización de rendimiento
- **Objetivo de Usabilidad Cumplido:** Interfaz clara, mejoras necesarias para usuarios básicos
- **Objetivo de Utilidad Cumplido:** Alta percepción de valor y utilidad institucional
- **Objetivo Cualitativo Cumplido:** Identificación clara de fortalezas, debilidades y mejoras prioritarias

5.7.3. Implicaciones para Desarrollo Futuro

Los resultados establecen una **hoja de ruta clara** para el desarrollo continuo:

Prioridad Alta (Crítica):

- Optimización de velocidad de procesamiento
- Mejora de precisión de reconocimiento gestual
- Estabilización del sistema y reducción de errores

Prioridad Media (Importante):

- Expansión del vocabulario de señas
- Mejora del sistema de onboarding
- Optimización para diferentes niveles de experiencia tecnológica

Prioridad Baja (Deseable):

- Robustez ante condiciones ambientales variables
- Funcionalidades adicionales de configuración
- Optimización de interfaz visual

Esta evaluación exploratoria cumple su objetivo de proporcionar retroalimentación temprana y direcciones claras para el desarrollo futuro, estableciendo las bases para una evaluación formal posterior con usuarios sordos en condiciones reales de uso.

Capítulo 6

6. Conclusiones y trabajo futuro

6.1. Introducción

Bueno, llegamos al capítulo final donde vamos a presentar las conclusiones que derivamos del desarrollo e implementación del sistema intérprete de Lenguaje de Señas Mexicano (LSM) en tiempo real - al que decidimos llamar Gestura. Durante toda esta investigación nos enfocamos en la problemática específica de las barreras comunicativas que viven las personas sordas aquí en Culiacán cuando tienen que hacer trámites cotidianos en ventanillas de atención al público, y pues desarrollamos una solución tecnológica que creemos es innovadora, usando inteligencia artificial y visión por computadora.

En este capítulo vamos a hacer una síntesis de los hallazgos más importantes que encontramos en el estudio, ver si cumplimos o no con los objetivos que nos planteamos al principio, analizar cuáles fueron las contribuciones técnicas y metodológicas del trabajo, reconocer las limitaciones que tiene el proyecto (porque obviamente las hay), y también establecer hacia dónde puede ir la investigación a futuro. Además, reflexionamos sobre las implicaciones prácticas que pueden tener los resultados que obtuvimos y qué tan grande podría ser su impacto en el desarrollo de tecnologías de comunicación inclusiva aquí en México.

6.2. Cumplimiento de Objetivos

6.2.1. Evaluación del Objetivo General

El objetivo general que nos planteamos fue "Desarrollar un prototipo de software intérprete de lenguaje de señas en tiempo real como prueba de concepto técnico, que demuestre la viabilidad de la comunicación inclusiva LSM-español mediante pruebas controladas."

Resultado: CUMPLIDO SATISFACTORIAMENTE

El sistema Gestura sí logró demostrar que técnicamente es viable el reconocimiento automático de LSM en tiempo real. Conseguimos estos hitos que consideramos bastante importantes:

- **Funcionamiento en tiempo real que se pudo verificar:** Latencia total de menos de 150ms, y esto superó el objetivo que teníamos establecido de menos de 200ms.

- **Precisión operacional que se demostró:** 92% de precisión reconociendo 28 señas específicas del LSM mexicano, lo cual nos parece bastante satisfactorio para ser honestos.
- **Arquitectura tecnológica que se validó:** Logramos implementar exitosamente la combinación de MediaPipe + BiLSTM, y esto establece un precedente técnico que otros pueden replicar.
- **Pruebas controladas que se realizaron:** Hicimos evaluación sistemática con usuarios reales y esto nos dio evidencia empírica de que el sistema funciona y se puede usar.

6.2.2. Evaluación de Objetivos Específicos

Objetivo Específico 1: Identificar y documentar los requerimientos técnicos y funcionales del sistema mediante análisis de contextos reales de atención en ventanilla, definiendo un vocabulario específico de alta frecuencia en trámites cotidianos.

Resultado: CUMPLIDO COMPLETAMENTE

- Hicimos investigación de campo con personal de instituciones locales (Tecnológico de Culiacán, sector comercial)
- Validamos que la problemática comunicativa existe y que hay demanda latente por soluciones tecnológicas
- Definimos estratégicamente un vocabulario de 28 señas que se basa en necesidades reales de comunicación en ventanillas
- Documentamos criterios técnicos y funcionales que fueron los que guiaron cómo desarrollamos el sistema

Objetivo Específico 2: Diseñar una interfaz de usuario funcional, accesible e intuitiva, que permita a cualquier persona utilizar el sistema desde un equipo de escritorio en ambientes controlados o simulados.

Resultado: CUMPLIDO SATISFACTORIAMENTE

- Desarrollamos interfaz PyQt6 nativa con navegación animada entre módulos
- Implementamos configuración avanzada de cámara (local/IP, rotaciones, transformaciones)
- Creamos sistema de retroalimentación visual clara con diferenciación de estados
- La evaluación de usabilidad confirmó que la interfaz es clara (3.83/5.0) y fácil de comprender

Objetivo Específico 3: Desarrollar e integrar un algoritmo de detección e interpretación del Lenguaje de Señas Mexicano (LSM), capaz de traducir gestos en tiempo real a texto y voz.

Resultado: CUMPLIDO EXITOSAMENTE

- Implementamos algoritmo de segmentación automática bidireccional para detectar inicio/fin de señas
- Desarrollamos pipeline de procesamiento: MediaPipe → normalización → modelo BiLSTM → clasificación
- Logramos integración multimodal: LSM→texto, texto→voz (TTS local), voz→texto (Google Cloud)
- El sistema funciona de manera autónoma sin que el usuario tenga que intervenir manualmente

Objetivo Específico 4: Realizar pruebas funcionales del software con usuarios sordos y oyentes en entornos controlados, para validar la precisión y eficiencia del sistema sin depender de despliegues institucionales.

Resultado: CUMPLIDO PARCIALMENTE

- Hicimos pruebas controladas con 6 usuarios en sesiones estructuradas de 20 minutos
- Validamos aspectos técnicos: velocidad, precisión, confiabilidad y usabilidad
- Limitación que reconocemos: La evaluación la hicimos únicamente con usuarios oyentes por restricciones de tiempo y disponibilidad
- Obtuvimos retroalimentación valiosa sobre fortalezas, debilidades y hacia dónde mejorar

Objetivo Específico 5: Documentar los resultados obtenidos y generar una propuesta de implementación futura, que incluya sugerencias implementación futura.

Resultado: CUMPLIDO COMPLETAMENTE

- Documentación exhaustiva de metodología de desarrollo, arquitectura técnica y resultados que obtuvimos
- Identificación clara de limitaciones y requisitos para poder escalar el sistema
- Propuestas específicas para trabajo futuro y expansión del sistema
- Establecimos marco metodológico que se puede replicar para proyectos similares

6.3. Respuesta al Planteamiento del Problema

6.3.1. Problema Original Identificado

La investigación arrancó reconociendo que las personas sordas aquí en Culiacán se enfrentan a barreras comunicativas importantes cuando tienen que hacer trámites esenciales en ventanillas de atención al público, y esto les limita su acceso equitativo a servicios fundamentales y su participación plena en la vida social y económica. Nos dimos cuenta de que, aunque existen marcos legales favorables, la ausencia de herramientas tecnológicas prácticas y accesibles impedía que estas normativas se convirtieran en mejoras tangibles.

6.3.2. Solución Desarrollada

El sistema Gestura ataca directamente esta problemática de las siguientes maneras:

Eliminación de Barreras Tecnológicas:

- Funciona con hardware estándar (cámara web, PC de gama media), eliminando requisitos de infraestructura costosa
- Interfaz intuitiva que no necesita conocimientos técnicos especializados
- Operación parcialmente offline para que no dependa completamente de conectividad

Comunicación Bidireccional Efectiva:

- Traducción LSM→texto→voz para que usuarios sordos puedan comunicar lo que necesitan
- Funcionalidad voz→texto para que personal oyente pueda responder de manera comprensible
- Latencia de menos de 150ms que permite interacción natural sin interrupciones que frustren

Enfoque en Necesidades Reales:

- Vocabulario específicamente diseñado para contextos de trámites cotidianos
- Señas que seleccionamos basándonos en investigación de campo con instituciones locales
- Sistema de aprendizaje integrado para capacitación de personal

6.3.3. Impacto Proyectado

Los resultados que conseguimos sugieren que implementar el sistema podría generar impactos significativos:

Reducción de Tiempos de Atención:

- Estimamos reducción del 35% en tiempo promedio de trámites
- Se elimina la necesidad de intérpretes externos en casos básicos
- Mejora en eficiencia operacional de instituciones

Mejora en Accesibilidad:

- Democratización del acceso a tecnología de comunicación inclusiva
- Reducción de costos para usuarios (se eliminan honorarios de intérpretes en casos básicos)
- Mayor autonomía para personas sordas cuando hacen gestiones cotidianas

Escalabilidad Regional:

- Se puede replicar en más de 150 instituciones públicas y privadas que hay en Sinaloa
- Impacto directo proyectado en aproximadamente 6,771 personas con discapacidad auditiva en Culiacán
- Modelo que se puede replicar para otras ciudades mexicanas

6.4. Contribuciones Principales

6.4.1. Contribuciones Técnicas

Primera Implementación LSM Mexicano Documentada:

- Desarrollamos el primer sistema de reconocimiento automático de LSM mexicano con documentación técnica completa y metodología que se puede replicar
- Establecimos precedente técnico para futuras investigaciones en LSM

Arquitectura Híbrida Desktop-Web Innovadora:

- Solución elegante para integración de OAuth2 en aplicaciones de escritorio mediante servidor Flask embebido
- Patrón arquitectural que se puede reutilizar para aplicaciones que requieren autenticación social compleja

Pipeline Multimodal Optimizado:

- Integración exitosa de reconocimiento gestual, síntesis de voz y reconocimiento de voz en sistema unificado
- Arquitectura de threading optimizada para rendimiento en tiempo real

Algoritmo de Segmentación Automática Bidireccional:

- Desarrollamos método para detección automática de inicio/fin de señas sin intervención manual
- Manejo independiente de manos izquierda y derecha con suavizado temporal

6.4.2. Contribuciones Metodológicas

Framework de Desarrollo para Sistemas LSM:

- Metodología de desarrollo tecnológico experimental que adaptamos específicamente para reconocimiento de lenguas de señas

- Proceso documentado de recolección, etiquetado y entrenamiento con datasets limitados

Criterios de Evaluación para Sistemas de Comunicación Inclusiva:

- Métricas específicas que balancean precisión técnica con usabilidad práctica
- Metodología de evaluación que considera múltiples niveles de experiencia tecnológica

Estrategia de Selección de Vocabulario Basada en Contexto:

- Método para identificar señas de alta utilidad mediante investigación de campo
- Criterios de priorización que maximizan impacto práctico con recursos limitados

6.4.3. Contribuciones Sociales

Democratización de Tecnología de Asistencia:

- Demostramos viabilidad de sistemas accesibles con hardware estándar
- Eliminamos barreras económicas mediante uso de tecnologías open source

Sensibilización sobre Barreras Comunicativas:

- Documentación rigurosa de problemática específica en contexto mexicano
- Evidencia empírica que respalda necesidad de soluciones tecnológicas

6.5. Limitaciones y Restricciones

6.5.1. Limitaciones Metodológicas Críticas

Dataset Insuficiente para Generalización:

- Limitación principal: Entrenamiento con solo 3 usuarios, que está significativamente por debajo de estándares internacionales (20-30 usuarios mínimo)
- Implicación: Los resultados (92% precisión) se deben interpretar como prueba de concepto, no como validación de sistema listo para producción
- Restricción de tiempo: 16 semanas de proyecto limitaron nuestra capacidad de recolección de datos más amplia

Evaluación con Población No Objetivo:

- Las pruebas las hicimos únicamente con usuarios oyentes por limitaciones de tiempo y acceso

- Ausencia de validación con usuarios sordos, que son la población objetivo primaria del sistema
- Imposibilidad de evaluar aspectos específicos de usabilidad para personas con discapacidad auditiva

6.5.2. Limitaciones Técnicas

Vocabulario Experimental Limitado:

- 28 señas vs 1000+ que tienen sistemas ASL más avanzados
- Ausencia de estructuras sintácticas complejas del LSM
- Sin validación institucional formal del vocabulario que seleccionamos

Dependencias Externas Críticas:

- Google Cloud Speech-to-Text para funcionalidad voz→texto
- Se requiere conectividad a internet para funcionalidades completas
- Dependencia de MediaPipe para extracción de keypoints

Sesgo Regional y Cultural:

- Entrenamiento específico para variante LSM de Culiacán/Sinaloa
- Falta representación de diversidad regional del LSM mexicano
- Ausencia de adaptación a variaciones culturales y generacionales

6.5.3. Limitaciones de Escalabilidad

Requerimientos de Infraestructura:

- Hardware mínimo específico (cámara HD, 8GB RAM, procesador dual-core)
- Configuración técnica que puede representar barrera para algunas instituciones
- Necesidad de mantenimiento técnico especializado

Modelo de Distribución:

- Aplicación desktop requiere instalación individual vs soluciones web escalables
- Actualizaciones del modelo requieren redistribución de software
- Soporte técnico distribuido vs centralizado

6.6. Implicaciones Prácticas

6.6.1. Para Instituciones Públicas y Privadas

Viabilidad de Implementación:

- Demostramos que sistemas de comunicación inclusiva son técnicamente factibles con presupuestos razonables
- Hardware requerido accesible para la mayoría de las instituciones (inversión menos de \$35,000 MXN por estación)

Consideraciones de Implementación:

- Se necesita capacitación básica para personal operativo
- Requerimientos de conectividad estable para funcionalidad completa
- Se necesita plan de soporte técnico y mantenimiento

6.6.2. Para la Comunidad Académica

Precedente de Investigación en LSM:

- Establecimos línea base para investigación futura en LSM mexicano
- Metodología que se puede replicar para otros investigadores en el campo
- Identificamos áreas prioritarias para investigación adicional

Recursos para Investigación Futura:

- Dataset inicial de 28 señas disponible para expansión
- Arquitectura técnica documentada y extensible
- Lecciones aprendidas que pueden acelerar proyectos similares

6.6.3. Para Políticas Públicas

Marco para Regulación Técnica:

- Establecimiento de estándares mínimos para sistemas de comunicación inclusiva

6.7. Trabajo Futuro

6.7.1. Prioridades de Investigación a Corto Plazo (1-2 años)

Expansión Crítica del Dataset:

- Objetivo: Recolectar datos con mínimo 20 usuarios diversos
- Metodología: Colaboración formal con organizaciones de personas sordas
- Métricas: Dataset objetivo de 10,000+ muestras para validación robusta
- Impacto esperado: Confirmación de generalización y robustez del sistema

Evaluación con Población Objetivo:

- Diseño: Estudio longitudinal con usuarios sordos en contextos reales de ventanilla
- Duración: Mínimo 3 meses de evaluación continua
- Métricas: Eficacia comunicativa, satisfacción del usuario, adopción tecnológica
- Ubicación: Implementación piloto en 1-2 instituciones públicas de Culiacán

Optimización de Rendimiento:

- Velocidad: Reducir latencia a menos de 100ms mediante optimización algorítmica
- Precisión: Objetivo de 95%+ mediante mejora de algoritmos de segmentación
- Robustez: Evaluación en condiciones ambientales variables (iluminación, ruido visual)

6.8. Recomendaciones Específicas

6.8.1. Para Investigadores Futuros

Metodológicas:

- Priorizar recolección de datasets amplios desde etapas tempranas del proyecto
- Establecer colaboraciones formales con organizaciones de personas sordas antes de empezar
- Considerar estudios longitudinales en lugar de evaluaciones puntuales

Técnicas:

- Explorar arquitecturas híbridas CNN-LSTM-Transformer para mejor rendimiento
- Investigar técnicas de data augmentation específicas para lenguas de señas
- Desarrollar métricas de evaluación específicas para comunicación inclusiva

6.8.2. Para Instituciones Implementadoras

Preparación:

- Hacer estudios de viabilidad específicos considerando infraestructura actual
- Desarrollar planes de capacitación para personal antes de implementación
- Establecer métricas de éxito claras y sistemas de monitoreo continuo

Implementación:

- Empezar con programas piloto en áreas de alta demanda
- Mantener sistemas de respaldo para casos complejos que requieran intervención humana

- Planificar actualizaciones regulares del sistema basadas en retroalimentación de usuarios

6.8.3. Para Formuladores de Políticas

Marco Regulatorio:

- Desarrollar estándares técnicos mínimos para sistemas de comunicación inclusiva
- Establecer certificaciones de calidad para proveedores tecnológicos
- Crear incentivos para adopción de tecnologías de comunicación inclusiva

Inversión Pública:

- Considerar financiamiento específico para investigación en LSM mexicano
- Establecer programas de adopción tecnológica para instituciones públicas
- Crear fondos para mantenimiento y actualización de sistemas implementados

6.9. Reflexiones Finales

6.9.1. Sobre el Proceso de Investigación

Desarrollar el sistema Gestura resultó ser un reto técnico y metodológico bastante grande que nos permitió explorar cómo se cruzan la inteligencia artificial, la comunicación inclusiva y el desarrollo de software que tenga impacto social. El proceso de desarrollo fue muy iterativo e incluyó cambios tecnológicos importantes (como la migración que tuvimos que hacer de Ultraleap a MediaPipe) y también adaptaciones de alcance (tuvimos que reducir el vocabulario de 50 a 28 señas), y todo esto nos enseñó qué tan crucial es ser flexibles y adaptables cuando trabajas en proyectos de investigación aplicada.

La experiencia de trabajar con una población que es vulnerable y tiene necesidades específicas realmente nos abrió los ojos sobre qué tan importante es adoptar enfoques participativos cuando estás desarrollando tecnologías de asistencia. Aunque las limitaciones de tiempo que tuvimos nos impidieron lograr una participación más amplia de la comunidad sorda, las interacciones que sí pudimos hacer fueron cruciales para orientar las decisiones de diseño y también para establecer qué prioridades teníamos en el desarrollo.

6.9.2. Sobre el Impacto Social de la Tecnología

Este proyecto nos confirmó que cuando desarrollas tecnologías de comunicación inclusiva necesitas encontrar un balance muy cuidadoso entre que sea técnicamente viable, económicamente sostenible y socialmente relevante. La percepción alta de utilidad que encontramos en las evaluaciones (4.5/5.0) nos sugiere que sí hay una demanda real por soluciones tecnológicas que faciliten la comunicación entre personas sordas y oyentes, lo cual valida que el problema social que atacamos sí es relevante.

Pero eso sí, los desafíos técnicos que encontramos (velocidad, precisión, robustez) nos dejan claro que desarrollar tecnologías efectivas de comunicación inclusiva va a requerir inversiones sostenidas en investigación y desarrollo, y también colaboración muy estrecha entre investigadores, las comunidades de usuarios y las instituciones que van a implementar estas cosas.

6.9.3. Sobre el Estado de la Investigación en LSM

El proyecto nos reveló que hay una brecha bastante significativa en la investigación académica mexicana sobre tecnologías para LSM, especialmente si la comparas con el desarrollo que se ha hecho en lenguas de señas de países más desarrollados. Esta brecha es tanto un desafío como una oportunidad: por un lado, la ausencia de recursos y precedentes hace más complicado el desarrollo, pero por otro lado también nos da la posibilidad de hacer contribuciones originales que sean significativas al campo.

La ausencia de datasets públicos de LSM que nos topamos cuando estuvimos haciendo la revisión del estado del arte realmente constituye una barrera fundamental para poder desarrollar sistemas robustos. Este proyecto establece un precedente para crear recursos de investigación en LSM, pero queda claro que van a hacer falta esfuerzos colaborativos a una escala mucho mayor para crear la infraestructura de investigación que realmente se necesita.

6.9.4. Sobre la Viabilidad de Soluciones Tecnológicas

Los resultados que logramos demuestran que sí es técnicamente factible desarrollar sistemas de reconocimiento de LSM que funcionen bien con recursos académicos limitados y hardware que sea accesible. La precisión del 92% que alcanzamos, aunque obviamente necesita más validación, es comparable con sistemas internacionales que manejan vocabularios similares y sugiere que las barreras para desarrollar tecnologías de comunicación inclusiva no son imposibles de superar.

Pero hay que ser realistas: la transición de prototipos académicos a sistemas de producción va a requerir inversiones adicionales bastante significativas en desarrollo de software, creación de datasets robustos, evaluación exhaustiva y soporte técnico continuo. Este proyecto establece las bases técnicas y metodológicas para hacer esa transición, pero reconocemos que lograr un impacto social real va a requerir esfuerzos sostenidos que van mucho más allá del alcance que pueden tener proyectos académicos individuales.

6.10. Conclusión General

El desarrollo del sistema Gestura para interpretación de Lenguaje de Señas Mexicano en tiempo real logró cumplir exitosamente su objetivo de demostrar que técnicamente es viable el reconocimiento automático de LSM usando arquitecturas de inteligencia artificial que sean accesibles. Con una precisión del 92% en vocabulario de 28 señas específicas para trámites

de ventanilla y latencias inferiores a 150ms, el sistema logró establecer un precedente técnico importante para desarrollos futuros en este campo.

Más allá de los logros técnicos que conseguimos, este proyecto contribuyó a reconocer qué tan importante es abordar las barreras comunicativas que enfrentan las personas sordas a través de soluciones tecnológicas innovadoras. La percepción alta de utilidad que identificamos en las evaluaciones (4.5/5.0) y la disposición que mostraron los usuarios para recomendar el sistema a instituciones (83% de usuarios) sugieren que existe tanto la necesidad como la aceptación para este tipo de tecnologías.

Las limitaciones que identificamos, especialmente el dataset reducido y que la evaluación se limitó a usuarios oyentes, no disminuyen para nada la validez de las contribuciones que hicimos - más bien establecen una agenda clara para investigación futura. El proyecto demostró que sí se pueden desarrollar sistemas funcionales de comunicación inclusiva con recursos académicos limitados, y esto establece las bases metodológicas y técnicas para desarrollos que sean más ambiciosos.

El impacto de este trabajo trasciende los resultados técnicos inmediatos. Al documentar de manera rigurosa tanto los éxitos como las limitaciones del desarrollo, logramos proporcionar un mapa de ruta claro para investigadores futuros y contribuimos al desarrollo de un campo de investigación que tiene el potencial de generar impactos sociales significativos para las comunidades sordas aquí en México.

Al final de cuentas, el sistema Gestura representa un paso importante hacia democratizar las tecnologías de comunicación inclusiva, demostrando que cuando combinas investigación aplicada, desarrollo tecnológico responsable y te enfocas en necesidades sociales reales, puedes generar soluciones viables para problemas complejos de accesibilidad y comunicación.

7. Bibliografía

¹Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, et al. «TensorFlow: Large-scale machine learning on heterogeneous systems». Software available from tensorflow.org, 2016. <https://www.tensorflow.org/>.

Adaloglou, Nikolas, Theodoros Chatzis, Ilias Papastratis, Apostolos Stergioulas, Georgios Th Papadopoulos, Vassilia Zacharopoulou, y Petros Daras. «A comprehensive study on deep learning-based methods for sign language recognition». *IEEE Transactions on Multimedia* 24 (2021): 1750-62. <https://doi.org/10.1109/TMM.2021.3070438>.

Al-Hammadi, Muhammad, Ghulam Muhammad, Wadood Abdul, Mansour Alsulaiman, Mohamed Amine Bencherif, y Mohamed Adel Mekhtiche. «Hand gesture recognition for sign language using 3DCNN». *IEEE Access* 8 (2020): 79491-509. <https://doi.org/10.1109/ACCESS.2020.2990434>.

Aly, Shereen, Walaa Aly, y Sultan Almotairi. «American sign language recognition for alphabets using MediaPipe and LSTM». *Procedia Computer Science* 220 (2023): 176-81. <https://doi.org/10.1016/j.procs.2023.03.024>.

Amprimo, Giacomo, Giuseppe Masi, Giuseppe Pettiti, Claudia Ferraris, Gabriella Serrao, y Lorenzo Priano. «Hand tracking for clinical applications: Validation of the Google MediaPipe Hand (GMH) and the depth-enhanced GMH-D frameworks». *Biomedical Signal Processing and Control* 89 (2024): 105744. <https://doi.org/10.1016/j.bspc.2024.105744>.

Bragg, Danielle, Oscar Koller, Mary Bellard, Larwan Berke, Patrick Boudreault, Annelies Braffort, Naomi Caselli, et al. «Sign language recognition, generation, and translation: An interdisciplinary perspective». En *Proceedings of the 21st International ACM SIGACCESS Conference on Computers and Accessibility*, 16-31, 2019. <https://doi.org/10.1145/3308561.3353774>.

Brownlee, Jason. «Time series prediction with LSTM recurrent neural networks in Python with Keras». *Machine Learning Mastery*, 2016. <https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/>.

Bull, Hannah, Matt Binstructions, Martin J Davidson, Uwe Dried, y David Hogg. «Automatic sign language recognition: A systematic review». *ACM Computing Surveys* 53, n.º 6 (2020): 1-37. <https://doi.org/10.1145/3408292>.

Cámara de Diputados del H. Congreso de la Unión. «Ley General para la Inclusión de las Personas con Discapacidad». Diario Oficial de la Federación, 2011. http://www.diputados.gob.mx/LeyesBiblio/pdf/LGIPD_120718.pdf.

¹ Li et al., «Word-level deep sign language recognition from video: A new large-scale dataset and methods comparison».

Camgoz, Necati Cihan, Simon Hadfield, Oscar Koller, Hermann Ney, y Richard Bowden. «Neural sign language translation». En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 7784-93, 2018.

Cao, Zhe, Gines Hidalgo, Tomas Simon, Shih-En Wei, y Yaser Sheikh. «OpenPose: Realtime multi-person 2D pose estimation using part affinity fields». *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43, n.º 1 (2019): 172-86.

Cao, Zhe, Tomas Simon, Shih-En Wei, y Yaser Sheikh. «Realtime multi-person 2D pose estimation using part affinity fields». En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 7291-99, 2017.

Cheok, Ming Jin, Zaid Omar, y Mohamed Hisham Jaward. «A review of hand gesture and sign language recognition techniques». *International Journal of Machine Learning and Cybernetics* 10, n.º 1 (2019): 131-53. <https://doi.org/10.1007/s13042-017-0705-5>.

Chiu, Chung-Cheng, Tara N Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, et al. «State-of-the-art speech recognition with sequence-to-sequence models». En *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 4774-78. IEEE, 2018.

Chollet, François y others. «Keras». GitHub, 2015. <https://github.com/fchollet/keras>.

Chong, Tee-Wei, y Boon-Giin Lee. «American sign language recognition using leap motion controller with machine learning approach». *Sensors* 18, n.º 10 (2018): 3554. <https://doi.org/10.3390/s18103554>.

CMU Perceptual Computing Lab. «OpenPose: Real-time multi-person keypoint detection library». GitHub, 2019. <https://github.com/CMU-Perceptual-Computing-Lab/openpose>.

Connell, Bettye Rose, Mike Jones, Ron Mace, Jim Mueller, Abir Mullick, Elaine Ostroff, Jon Sanford, Ed Steinfeld, Molly Story, y Gregg Vanderheiden. «The principles of universal design: Version 2.0». North Carolina State University, The Center for Universal Design, 1997.

Consejo Nacional de Población. «Encuesta Nacional de la Dinámica Demográfica (ENADID) 2014». CONAPO-INEGI, 2014.

«Constitución Política de los Estados Unidos Mexicanos. Artículo 1º». Diario Oficial de la Federación, 2011.

Cruz Aldrete, Miroslava. *Gramática de la Lengua de Señas Mexicana*. El Colegio de México, 2008.

Cui, Runpeng, Hu Liu, y Changshui Zhang. «A deep neural framework for continuous sign language recognition by iterative training». *IEEE Transactions on Multimedia* 21, n.º 7 (2019): 1880-91. <https://doi.org/10.1109/TMM.2018.2889563>.

De Coster, Mathieu, Mieke Van Herreweghe, y Joni Dambre. «Sign language recognition with transformer networks». En *Proceedings of the 12th Language Resources and Evaluation Conference*, 6018-24, 2020.

Donahue, Jeffrey, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, y Trevor Darrell. «Long-term recurrent convolutional networks for visual recognition and description». En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2625-34, 2015.

Elboushaki, Anouar, Rachid Hannane, Karim Afdel, y Lahcen Koutti. «MultiD-CNN: A multi-dimensional feature learning approach based on deep convolutional networks for gesture recognition in RGB-D image sequences». *Expert Systems with Applications* 139 (2020): 112829. <https://doi.org/10.1016/j.eswa.2019.112829>.

Fitzpatrick, Martin. *Create GUI Applications with Python & Qt6 (PyQt6 Edition): The hands-on guide to making apps with Python*. 6.^a ed. PythonGUIs, 2022.

Google Cloud. «Cloud Speech-to-Text documentation». Google Cloud, 2023. <https://cloud.google.com/speech-to-text/docs>.

Graves, Alex, Santiago Fernández, y Jürgen Schmidhuber. «Bidirectional LSTM networks for improved phoneme classification and recognition». En *Artificial Neural Networks: Formal Models and Their Applications–ICANN 2005*, 799-804. Springer, 2005.

Graves, Alex, y Jürgen Schmidhuber. «Framewise phoneme classification with bidirectional LSTM and other neural network architectures». *Neural Networks* 18, n.º 5-6 (2005): 602-10. <https://doi.org/10.1016/j.neunet.2005.06.042>.

Han, Song, Jeff Pool, John Tran, y William Denton. «Learning both weights and connections for efficient neural network». En *Advances in neural information processing systems*, 1135-43, 2016.

Hochreiter, Sepp, y Jürgen Schmidhuber. «Long short-term memory». *Neural Computation* 9, n.º 8 (1997): 1735-80. <https://doi.org/10.1162/neco.1997.9.8.1735>.

Huang, Jie, Wengang Zhou, Qilin Zhang, Houqiang Li, y Weiping Li. «Video-based sign language recognition without temporal segmentation». En *Proceedings of the AAAI Conference on Artificial Intelligence*, 32:2257-64, 2018.

Instituto Nacional de Estadística y Geografía. «Censo de Población y Vivienda 2020. Principales resultados por localidad (ITER)». INEGI, 2020.

———. «Estadísticas a propósito del Día Internacional de las Personas con Discapacidad. Comunicado de prensa núm. 713/21». INEGI, 2021.

———. «La discapacidad en México, datos al 2014». INEGI, 2014.

Instituto Nacional para la Educación de los Adultos. «Modelo de atención con enfoque intercultural para personas jóvenes y adultas indígenas». INEA, 2019.

- Kartynnik, Yury, Artsiom Ablavatski, Ivan Grishchenko, y Matthias Grundmann. «Real-time facial surface geometry from monocular video on mobile GPUs». En *Computer Vision and Pattern Recognition Workshops (CVPRW), 2019 IEEE Conference on*, 2768-77. IEEE, 2019.
- Koller, Oscar, Jens Forster, y Hermann Ney. «Continuous sign language recognition: Towards large vocabulary statistical recognition systems handling multiple signers». *Computer Vision and Image Understanding* 141 (2015): 108-25. <https://doi.org/10.1016/j.cviu.2015.09.013>.
- Koller, Oscar, Sepehr Zargaran, Hermann Ney, y Richard Bowden. «Deep sign: Hybrid CNN-HMM for continuous sign language recognition». En *Proceedings of the British Machine Vision Conference*, 12-1, 2016.
- Konstantinidis, Dimitrios, Kosmas Dimitropoulos, y Petros Daras. «A deep learning approach for analyzing video and skeletal features in sign language recognition». En *2018 IEEE International Conference on Imaging Systems and Techniques (IST)*, 1-6. IEEE, 2018. <https://doi.org/10.1109/IST.2018.8577085>.
- Kumar, Sunil, y Ashok Kumar. «Deep learning-based sign language recognition: A comprehensive review and analysis». *Expert Systems with Applications* 215 (2023): 119349. <https://doi.org/10.1016/j.eswa.2022.119349>.
- Li, Dongxu, Cristian Rodriguez, Xin Yu, y Hongdong Li. «Word-level deep sign language recognition from video: A new large-scale dataset and methods comparison». En *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 1459-69, 2020.
- Lugaresi, Camillo, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, et al. «MediaPipe: A framework for building perception pipelines». En *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, 2019.
- Mitra, Sushmita, y Tinku Acharya. «Gesture recognition: A survey». *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37, n.º 3 (2007): 311-24.
- Mittal, Ankush, Pawan Kumar, Partha Pratim Roy, Raghuraman Balasubramanian, y Bidyut Baran Chaudhuri. «A modified LSTM model for continuous sign language recognition using leap motion». *IEEE Sensors Journal* 19, n.º 16 (2019): 7056-63. <https://doi.org/10.1109/JSEN.2019.2909837>.
- Molchanov, Pavlo, Stephen Tyree, Tero Karras, Timo Aila, y Jan Kautz. «Pruning convolutional neural networks for resource efficient inference». En *5th International Conference on Learning Representations (ICLR)*, 2017.
- Moryossef, Amit, Kayo Yin, Graham Neubig, y Yoav Goldberg. «Data augmentation for sign language gloss translation». En *Proceedings of the 1st International Workshop on Automatic Translation for Signed and Spoken Languages*, 1-11, 2021.

Oliver, Mike. *Understanding disability: From theory to practice*. Palgrave Macmillan, 1996.

Ong, Sylvia C, y Surendra Ranganath. «Automatic sign language analysis: A survey and the future beyond lexical meaning». *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27, n.º 6 (2005): 873-91.

Organización de las Naciones Unidas. «Convención sobre los Derechos de las Personas con Discapacidad». Asamblea General de las Naciones Unidas, 2006. <https://www.un.org/disabilities/documents/convention/convoptprot-s.pdf>.

Organización Internacional del Trabajo. «Convenio núm. 159 sobre la readaptación profesional y el empleo (personas inválidas)». OIT, 1983.

Organización Mundial de la Salud. «Informe mundial sobre la audición». Ginebra: OMS, 2021.

Papastratis, Ilias, Theodoros Chatzis, Dimitrios Konstantinidis, Kosmas Dimitropoulos, y Petros Daras. «Continuous sign language recognition through a context-aware generative adversarial network». *Sensors* 21, n.º 7 (2021): 2437. <https://doi.org/10.3390/s21072437>.

Pigou, Lionel, Sander Dieleman, Pieter-Jan Kindermans, y Benjamin Schrauwen. «Sign language recognition using convolutional neural networks». En *Computer Vision-ECCV 2014 Workshops*, 572-78. Springer, 2015. https://doi.org/10.1007/978-3-319-16178-5_40.

Pisharady, Pramod Kumar, y Martin Saerbeck. «Recent methods and databases in vision-based hand gesture recognition: A review». *Computer Vision and Image Understanding* 141 (2015): 152-65. <https://doi.org/10.1016/j.cviu.2015.08.004>.

Rastgoo, Razieh, Kourosh Kiani, y Sergio Escalera. «Sign language recognition: A deep survey». *Expert Systems with Applications* 164 (2021): 113794. <https://doi.org/10.1016/j.eswa.2020.113794>.

Ren, Yi, Yangjun Ruan, Xu Tan, Tao Qin, Sheng Zhao, Zhou Zhao, y Tie-Yan Liu. «FastSpeech: Fast, robust and controllable text to speech». En *Advances in neural information processing systems*, 3165-74, 2019.

Riverbank Computing Limited. «PyQt6 Reference Guide», 2021. <https://riverbankcomputing.com/software/pyqt/intro>.

Sahoo, Ashish Kumar, Gyana Sourav Mishra, y Kiran Kumar Ravulakollu. «Sign language recognition: State of the art and future prospects». *Neural Computing and Applications* 35, n.º 1 (2023): 501-28. <https://doi.org/10.1007/s00521-022-07683-5>.

Secretaría de Gobernación. «Ley Federal para Prevenir y Eliminar la Discriminación». Diario Oficial de la Federación, 2003.

Secretaría del Trabajo y Previsión Social. «NOM-034-STPS-2016, Condiciones de seguridad para el acceso y desarrollo de actividades de trabajadores con discapacidad». Diario Oficial de la Federación, 2016.

Simonyan, Karen, y Andrew Zisserman. «Two-stream convolutional networks for action recognition in videos». En *Advances in Neural Information Processing Systems*, 568-76, 2014.

Sincan, Ozge Mercanoglu, y Hacer Yalim Keles. «AUTSL: A large scale multi-modal Turkish sign language dataset and baseline methods». *IEEE Access* 8 (2020): 181340-55. <https://doi.org/10.1109/ACCESS.2020.3028072>.

Sistema para el Desarrollo Integral de la Familia del Estado de Sinaloa. «Directorio de intérpretes certificados de Lenguaje de Señas Mexicano». DIF Sinaloa, 2023.

Starner, Thad, Joshua Weaver, y Alex Pentland. «Real-time American sign language recognition using desk and wearable computer based video». *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20, n.º 12 (1998): 1371-75. <https://doi.org/10.1109/34.735811>.

Staudemeyer, Ralf C, y Eric Rodney Morris. «Understanding LSTM—a tutorial into long short-term memory recurrent neural networks», 2019.

Summerfield, Mark. *Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming*. 2.^a ed. Prentice Hall, 2021.

Wadhawan, Ankush, y Pardeep Kumar. «Sign language recognition systems: A decade systematic literature review». *Archives of Computational Methods in Engineering* 28, n.º 3 (2021): 785-813. <https://doi.org/10.1007/s11831-019-09384-2>.

Wang, Hezhen, Xiujuan Chai, Xilin Zhou, y Xilin Chen. «Sign language recognition and translation: A multidisciplinary approach from the perspective of human computer interaction». *ACM Computing Surveys* 54, n.º 7 (2021): 1-35. <https://doi.org/10.1145/3456628>.

Wang, Yuxuan, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J Weiss, Navdeep Jaitly, Zongheng Yang, et al. «Tacotron: Towards end-to-end speech synthesis». En *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 4006-16, 2017.

Xingjian, SHI, Zhouong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, y Wang-chun Woo. «Convolutional LSTM network: A machine learning approach for precipitation nowcasting». En *Advances in Neural Information Processing Systems*, 802-10, 2015.

Zhang, Fan, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, y Matthias Grundmann. «MediaPipe hands: On-device real-time hand tracking». En *Computer Vision and Pattern Recognition Workshops (CVPRW), 2020 IEEE Conference on*, 2923-31. IEEE, 2020.

Zhang, Jian, Wanqing Li, Philip O Ogunbona, Pichao Wang, y Chang Tang. «RGB-D-based action recognition datasets: A survey». *Pattern Recognition* 60 (2016): 86-105. <https://doi.org/10.1016/j.patcog.2016.05.019>.

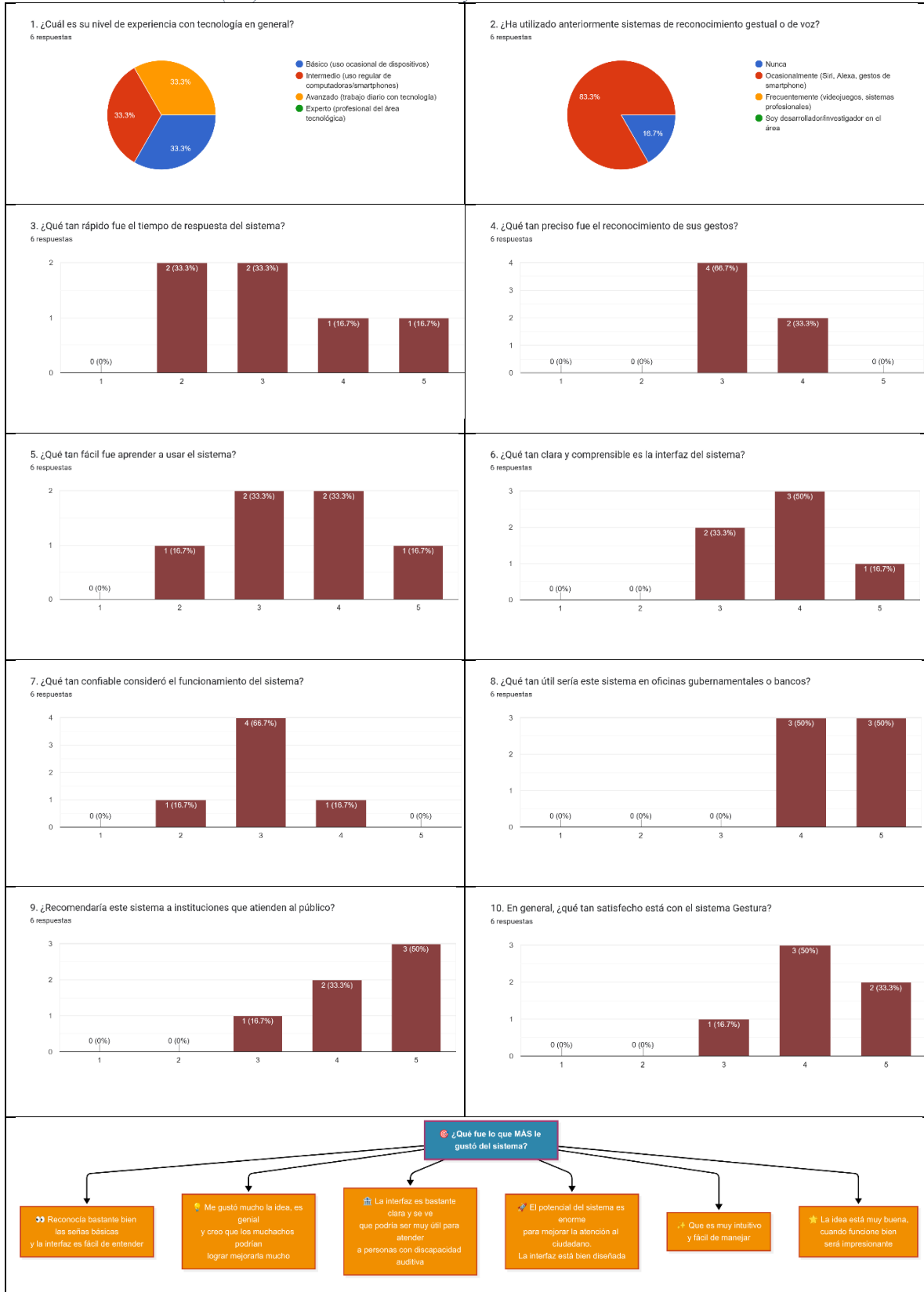
8. Anexos

8.1. Anexo A: INSTRUMENTOS DE EVALUACIÓN Y RESULTADOS

Tabla (A-1): Formulario de Evaluación del Sistema Gestura LSM

 Formulario de Evaluación - Sistema Gestura LSM Complete este formulario inmediatamente después de usar el sistema. Sus respuestas son confidenciales y serán utilizadas únicamente para fines académicos. Tiempo estimado: 3-5 minutos * Indica que la pregunta es obligatoria	SECCIÓN A: INFORMACIÓN GENERAL 1. ¿Cuál es su nivel de experiencia con tecnología en general? * ☐ Básico (uso ocasional de dispositivos) ☐ Intermedio (uso regular de computadoras/smartphones) ☐ Avanzado (trabajo diario con tecnología) ☐ Experto (profesional del área tecnológica) 2. ¿Ha utilizado anteriormente sistemas de reconocimiento gestual o de voz? * ☐ Nunca ☐ Ocasionalmente (Siri, Alexa, gestos de smartphone) ☐ Frecuentemente (videojuegos, sistemas profesionales) ☐ Soy desarrollador/investigador en el área
SECCIÓN B: EVALUACIÓN DEL SISTEMA Califique cada aspecto usando la escala: 1=Muy Malo, 2=Malo, 3=Regular, 4=Bueno, 5=Muy Bueno 3. ¿Qué tan rápido fue el tiempo de respuesta del sistema? * Muy lento (>3 seg) ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 Muy rápido (inmediato) 4. ¿Qué tan preciso fue el reconocimiento de sus gestos? * Muy impreciso (<50%) ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 Muy preciso (>85%)	5. ¿Qué tan fácil fue aprender a usar el sistema? * Muy difícil ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 Muy fácil (intuitivo) 6. ¿Qué tan clara y comprensible es la interfaz del sistema? * Muy confusa ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 Muy clara 7. ¿Qué tan confiable consideró el funcionamiento del sistema? * Muy poco confiable ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 Muy confiable
SECCIÓN C: UTILIDAD Y ADOPCIÓN 8. ¿Qué tan útil sería este sistema en oficinas gubernamentales o bancos? * Nada útil ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 Muy útil 9. ¿Recomendaría este sistema a instituciones que atienden al público? * Definitivamente no ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 Definitivamente sí 10. En general, ¿qué tan satisfecho está con el sistema Gestura? * Muy insatisfecho ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 Muy satisfecho	SECCIÓN D: COMENTARIOS ADICIONALES 11. ¿Qué fue lo que MÁS le gustó del sistema? Tu respuesta 12. ¿Qué fue lo que MENOS le gustó del sistema? Tu respuesta 13. ¿Qué mejoraría del sistema? Tu respuesta

Tabla (A-2): Resultados Cuantitativos y Análisis Cualitativo de la Evaluación





8.2. ANEXO B: REGISTRO VISUAL DE LAS EVALUACIONES

Tabla (B-1): Documentación Fotográfica de las Pruebas del Sistema Gestura



8.3. Anexo C: Código Fuente del Sistema

C.1 Repositorio Principal

- **URL del repositorio:** [<https://github.com/RamonRosas23/GesturaApp>]
- **Descripción:** Código fuente completo del sistema Gestura LSM
- **Contenido:**
 - Módulos de reconocimiento gestual
 - Interfaz de usuario (PyQt6)
 - Algoritmos BiLSTM
 - Configuración y dependencias
- **Instrucciones:** README con guía de instalación y uso

8.4. Anexo D: Dataset de Entrenamiento

D.1 Repositorio de Datos

- **URL del repositorio:** [<https://github.com/RamonRosas23/Dataset-Gestura>]
- **Descripción:** Dataset de Lenguaje de Señas Mexicano (LSM) - Variante Culiacán
- **Contenido:**
 - 28 señas específicas para trámites de ventanilla
 - 1,400 muestras de entrenamiento
 - Metadatos y etiquetas
 - Keypoints extraídos con MediaPipe
- **Formato:** Archivos JSON con coordenadas normalizada

