

**Prototipo de avatar traductor de voz y texto a lengua de señas colombiana para el
manejo de situaciones de emergencias**

**Laura Sofía Díaz Rodríguez
Luis Miguel Guerrero Pacanchique
Miguel Antonio López Cabra**

**Universidad de Boyacá
Facultad de Ciencias e Ingeniería
Ingeniería de Sistemas
Tunja
2024**

**Prototipo de avatar traductor de voz y texto a lengua de señas colombiana para el
manejo de situaciones de emergencias**

**Laura Sofía Díaz Rodríguez
Luis Miguel Guerrero Pacanchique
Miguel Antonio López Cabra**

**Trabajo de grado en semillero de investigación SIIAM, para optar al título de:
Ingeniero de Sistemas**

**Director
Carmen Constanza Uribe Sandoval
Doctora en Ingeniería con especialidad en Ingeniería de Sistemas**

**Universidad de Boyacá
Facultad de Ciencias e Ingeniería
Ingeniería de Sistemas
Tunja
2024**

Nota de aceptación:

Firma del Presidente del Jurado

Firma del Jurado

Tunja, 22 de mayo de 2024

“Únicamente el graduando es responsable de las ideas expuestas en el presente trabajo”.
(Lineamientos constitucionales, legales e institucionales que rigen la propiedad intelectual).

Contenido

	Pág.
Introducción.....	12
Exploración de técnicas de procesamiento de lenguaje natural para la traducción hacia la glosa de la lengua de señas colombiana.....	14
Investigación preliminar y retos iniciales	14
Evaluación de herramientas de NLP.....	14
Técnicas de tratamiento de lenguaje natural.....	15
Tokenización.....	15
Análisis morfosintáctico y semántico	15
Desarrollo del primer modelo de aprendizaje automático.....	16
Construcción y configuración del modelo.....	16
Recopilación de datos y preparación	17
Tokenización y secuenciación	17
Entrenamiento del modelo.....	18
Implementación de la función de traducción.....	18
Análisis de resultados	18
Tecnologías para la implementación de un avatar intérprete de lengua de señas colombiana a partir de texto en español.....	20
Primera etapa	20
Aplicaciones para el desarrollo front-end.....	21
Aplicaciones para el desarrollo back-end	22
Arquitectura de software.....	23
Comunicación cliente-servidor	26
Segunda etapa	26
Tercera etapa.....	28
Pruebas de desarrollo del modelo.....	31
Selección y utilización de la herramienta más adecuada para la creación del prototipo de avatar para señas de atención de emergencia en LSC con expresiones faciales y manuales ...	39
Análisis de herramientas.....	39

Elección de Daz Studio 3D.....	42
Desafíos de compatibilidad.....	42
Transición a Blender.....	43
Incorporación de Mixamo en Blender.....	43
Elaboración de las animaciones.....	43
Integración final del prototipo de avatar para la traducción de frases en español a la glosa de la lengua de señas colombiana.....	46
Conclusiones.....	48
Recomendaciones.....	51
Referencias.....	52
Anexos.....	54

Lista de Tablas

	Pág.
Tabla 1. Herramientas de diseño 3D	39

Lista de Figuras

	Pág.
Figura 1. Etiquetas de tokenización.....	29
Figura 2. Creación de la red transformer.....	31
Figura 3. Creación de diccionario de glosa	32
Figura 4. Instalación de dependencias.....	32
Figura 5. Importación de módulos y funciones	33
Figura 6. Preparación de los datos.....	33
Figura 7. Proceso de tokenización.....	34
Figura 8. Construcción de diccionario de tokens	35
Figura 9. Preprocesamiento de los datos	35
Figura 10. Mapeo de tokens a sus respectivos índices	36
Figura 11. Creación y ajuste de red neuronal	36
Figura 12. Entrenamiento de la red neuronal	37
Figura 13. Procesamiento de los datos	37
Figura 14. Resultados del entrenamiento	38
Figura 15. Animación en Daz Studio 3D	42
Figura 16. Animación GLB.....	44
Figura 17. Proceso de animación en Blender	45
Figura 18. Animación en Blender de palabra bien	45

Lista de Anexos

	Pág.
Anexo A. Anteproyecto	55
Anexo B. Corpus	76
Anexo C. Pruebas	77
Anexo D. Software	85

Resumen

Diseño de un prototipo de avatar intérprete de voz y texto a lengua de señas para dispositivos móviles:

El presente documento desarrolla un proyecto cuyo objetivo específico es identificar y aplicar técnicas avanzadas de procesamiento de lenguaje natural (NLP) para traducir texto y voz en español a la glosa de la lengua de señas colombiana (LSC). Además, se exploran y seleccionan tecnologías de programación para la implementación de un avatar que represente visualmente la LSC en situaciones de emergencia.

El objetivo general del trabajo es diseñar y desarrollar un prototipo de avatar intérprete que facilite la comunicación entre personas oyentes y no oyentes durante emergencias, mejorando así la inclusión y accesibilidad para personas con discapacidad auditiva.

La metodología utilizada incluye una investigación preliminar sobre NLP y técnicas de traducción a lenguas de señas, la recopilación y preparación de un corpus de datos en español y LSC, el desarrollo de un modelo de aprendizaje automático basado en redes neuronales recurrentes, y la implementación y evaluación de un avatar 3D usando herramientas como Daz Studio 3D y Blender.

Como conclusión general, el proyecto demuestra la viabilidad de utilizar tecnologías avanzadas para desarrollar herramientas que promuevan la inclusión de la comunidad sorda en situaciones críticas. Se recomienda continuar mejorando el prototipo y explorar nuevas tecnologías como los Transformers para mejorar la precisión de la traducción.

Palabras claves: Lengua de señas colombiana, procesamiento de lenguaje natural, avatar 3D, traducción automática, inclusión tecnológica, emergencia.

Abstract

Design of a Prototype Voice and Text Translator Avatar to Sign Language for Mobile Devices:

This document presents the development of a project with the specific objective of identifying and applying advanced natural language processing (NLP) techniques to translate text and voice in Spanish into the gloss of Colombian Sign Language (LSC). Additionally, programming technologies are explored and selected for implementing an avatar that visually represents LSC in emergency situations.

The general objective of the work is to design and develop a prototype interpreter avatar that facilitates communication between hearing and non-hearing individuals during emergencies, thereby improving inclusion and accessibility for people with hearing disabilities.

The methodology includes preliminary research on NLP and sign language translation techniques, the collection and preparation of a data corpus in Spanish and LSC, the development of a machine learning model based on recurrent neural networks, and the implementation and evaluation of a 3D avatar using tools such as Daz Studio 3D and Blender.

As a general conclusion, the project demonstrates the feasibility of using advanced technologies to develop tools that promote the inclusion of the deaf community in critical situations. It is recommended to continue improving the prototype and explore new technologies such as Transformers to enhance translation accuracy.

Keywords: Colombian Sign Language, natural language processing, 3D avatar, automatic translation, technological inclusion, emergency.

Introducción

La inclusión y la accesibilidad a la información son derechos fundamentales que deben estar garantizados para todas las personas, independientemente de su condición. Sin embargo, para las personas con discapacidad auditiva, esta inclusión suele verse obstaculizada por las barreras de comunicación existentes.

La lengua de señas se ha convertido en un medio de comunicación esencial para la comunidad sorda, permitiéndoles expresar sus pensamientos, emociones e ideas de manera fluida y natural. No obstante, el acceso limitado a intérpretes y recursos en lengua de señas puede dificultar su interacción con el mundo que los rodea, especialmente en situaciones de emergencia donde la comunicación efectiva es crucial.

En este contexto, la tecnología emerge como una herramienta poderosa para superar estas barreras y promover la inclusión de las personas con discapacidad auditiva. El presente proyecto propone el diseño de un prototipo de avatar intérprete de voz y texto a lengua de señas colombiana (LSC), específicamente enfocado en situaciones de emergencia. Este prototipo, desarrollado para dispositivos móviles Android, tiene como objetivo facilitar la comunicación efectiva entre personas oyentes y no oyentes en momentos en los que se requiera brindar ayuda ante situaciones adversas.

A través de técnicas avanzadas de procesamiento de lenguaje natural y animación 3D, el prototipo permitirá la traducción en tiempo real de preguntas claves en situaciones de emergencia a la glosa de la LSC para posteriormente ser representada visualmente por un avatar humanoide. Este enfoque innovador no solo promoverá la inclusión de la comunidad sorda en situaciones de emergencia, sino que también fomentará la conciencia y la comprensión de sus necesidades específicas en la sociedad en general.

El proyecto abarca diversas áreas de investigación, como el análisis de técnicas de procesamiento de lenguaje natural, la selección de herramientas de programación adecuadas, la creación y animación del avatar, y la evaluación del desempeño del prototipo en la traducción de textos y voz a la glosa de la LSC. Mediante un enfoque interdisciplinario y la aplicación de metodologías rigurosas, se busca contribuir al desarrollo de soluciones tecnológicas accesibles y eficientes que mejoren la calidad de vida de las personas con discapacidad auditiva.

En conclusión, este proyecto representa un paso significativo hacia la inclusión y la

accesibilidad en situaciones de emergencia, al brindar una herramienta innovadora que facilita la comunicación entre personas oyentes y no oyentes. Al abordar esta necesidad crucial, se espera fomentar una sociedad más inclusiva y equitativa, donde las barreras de comunicación no sean un obstáculo para la participación y el bienestar de todos.

Exploración de técnicas de procesamiento de lenguaje natural para la traducción hacia la glosa de la lengua de señas colombiana

El presente capítulo se enfoca en el desarrollo del objetivo específico: identificar las técnicas de procesamiento de lenguaje natural (NLP, por sus siglas en inglés) adecuadas para la traducción del español a la glosa de la lengua de señas colombiana (LSC). Este análisis abarca el estudio de diversas herramientas y metodologías de NLP, la evaluación de sus capacidades y limitaciones, y la selección de la técnica más eficiente para este propósito específico.

Investigación preliminar y retos iniciales

El primer paso en la investigación fue realizar una revisión de la literatura disponible sobre NLP y traducción de lenguas de señas. Este estudio permitió identificar las principales técnicas utilizadas en la traducción de lenguaje natural a lenguas de señas, tales como el análisis morfosintáctico, el procesamiento semántico, y la generación de lenguaje natural.

Uno de los retos más significativos fue la escasez de recursos y estudios previos específicamente centrados en la LSC. La mayoría de las investigaciones y herramientas disponibles se enfocan en lenguas de señas más difundidas, como la lengua de señas americana (ASL). Esto significó que gran parte del trabajo inicial involucró adaptar y probar técnicas generales para la lengua de señas colombiana.

Evaluación de herramientas de NLP

En la búsqueda de herramientas de NLP, se exploraron dos principales opciones. La primera fue Freeling, una plataforma de análisis de lenguaje ampliamente reconocida. A pesar de sus amplias capacidades en el análisis morfológico¹, sintáctico² y semántico del español, Freeling presentó limitaciones significativas para la tarea específica requerida. Principalmente, la herramienta no estaba optimizada para manejar la estructura y características únicas de la

¹ Implica descomponer palabras en sus partes más pequeñas para comprender su estructura gramatical y significado en un idioma

² Identificación de la estructura gramatical de una oración o texto.

glosa de la LSC, resultando en traducciones imprecisas y poco naturales.

Como alternativa, se consideró el uso de NLTK (Natural Language Toolkit), una biblioteca de procesamiento de lenguaje natural en Python, conocida por su flexibilidad y amplio rango de funcionalidades. Sin embargo, tras realizar su implementación, se observaron que los resultados obtenidos no cumplían con las expectativas deseadas.

Esta experiencia llevó a reconsiderar una aproximación y buscar alternativas como modelos de aprendizaje automático que son más flexibles y adaptables a las necesidades específicas del proyecto.

Técnicas de tratamiento de lenguaje natural

La solución del desafío vino de la mano de técnicas del procesamiento de lenguaje natural. Específicamente, en dos técnicas principales:

Tokenización

Esta técnica consiste en dividir el texto en unidades más pequeñas (tokens), lo cual es fundamental para entender la estructura del lenguaje de entrada. La tokenización en el contexto del español-LSC requirió un enfoque adaptado para manejar las diferencias estructurales entre las dos lenguas.

Análisis morfosintáctico y semántico

Esta técnica implica desglosar el texto en sus componentes básicos (palabras, frases) y entender su función y significado en el contexto. La correcta identificación de estas unidades y su relación es crucial para una traducción precisa a la glosa, que tiene una estructura gramatical diferente al español.

Desarrollo del primer modelo de aprendizaje automático

La base de cualquier sistema de NLP es un corpus³ de datos, por lo que se recopiló un conjunto de datos de textos en español y su correspondiente traducción en glosa⁴ de la LSC. Inicialmente, se recopiló 232 oraciones en español y sus correspondientes adaptaciones en glosa de la LSC, dichas oraciones fueron recolectadas del Diccionario Básico de la Lengua de Señas Colombiana. Estos datos se fueron almacenados en un archivo CSV, que posteriormente serviría como base para el entrenamiento y la evaluación del modelo

Construcción y configuración del modelo

Se optó por un modelo secuencial utilizando capas de redes neuronales recurrentes (LSTM), adecuadas para el procesamiento de secuencias. La arquitectura del modelo incluyó:

- Una capa de Embedding, que transforma los tokens de entrada en vectores densos de un tamaño especificado.
- Una capa LSTM para procesar la secuencia de entrada.
- Una RepeatVector para ajustar las dimensiones de salida.
- Otra capa LSTM para generar la secuencia de salida.
- Finalmente, una capa Dense con una función de activación 'softmax', para predecir la palabra siguiente en la glosa.

El modelo fue compilado con el optimizador 'adam' y la pérdida 'sparse_categorical_crossentropy', adecuados para problemas de clasificación multiclase como este. El modelo utilizado para la traducción del español a la glosa de la lengua de señas colombiana se basa en un enfoque de clasificación multiclase, donde cada palabra en español se clasifica en una de varias opciones en la glosa de la LSC. Además, el problema se considera multiclase debido a las múltiples opciones de traducción para cada palabra.

³ Colección de textos utilizada para investigar y analizar patrones lingüísticos en diferentes contextos y propósitos.

⁴ Representación visual y simplificada de conceptos o palabras clave utilizando gestos o signos específicos para facilitar la comunicación

El código utilizado en este proyecto se basó en una adaptación del repositorio disponible en GitHub llamado “Language Translation” del autor likarajo. Este repositorio proporcionó una base sólida para la implementación de sistemas de traducción automática utilizando TensorFlow y Keras. A partir de esta fuente, se realizaron ajustes y modificaciones para adecuar el código a las necesidades específicas del proyecto en cuestión, que se centra en la traducción del español a la glosa de la lengua de señas colombiana (LSC). Este enfoque permitió aprovechar la estructura y las técnicas establecidas en el repositorio mencionado, adaptándolas de manera eficiente para abordar los desafíos particulares de la traducción hacia la LSC.

Recopilación de datos y preparación

La base de cualquier sistema de NLP es un corpus de datos, por lo que se recopiló un conjunto de datos de textos en español y su correspondiente traducción en glosa de la LSC. Inicialmente, se recopiló 232 oraciones en español y sus correspondientes adaptaciones en glosa de la LSC, dichas oraciones fueron recolectadas del Diccionario Básico de la Lengua de Señas Colombiana. Estos datos se fueron almacenados en un archivo CSV, que posteriormente serviría como base para el entrenamiento y la evaluación del modelo.

Tokenización y secuenciación

El primer paso en el procesamiento de los datos fue la tokenización, que implica convertir las oraciones en español y las glosas en secuencias de tokens (números) que representan palabras individuales. Se utilizó el Tokenizer de Keras para esta tarea, creando dos tokenizadores distintos: uno para el español y otro para las glosas de la LSC. Esta tokenización es esencial para transformar el texto en un formato que las redes neuronales puedan procesar.

Después de la tokenización, se realizó un proceso de 'padding' (relleno) para asegurar que todas las secuencias tengan la misma longitud, lo cual es un requisito para el entrenamiento de modelos de redes neuronales.

Entrenamiento del modelo

El modelo se entrenó con el conjunto de datos de oraciones en español y sus correspondientes glosas en LSC, utilizando un total de 2500 épocas. Este proceso extenso fue necesario para permitir que el modelo aprendiera adecuadamente las complejas relaciones entre el español y la glosa de la LSC.

Implementación de la función de traducción

Tras el entrenamiento, se implementó una función para traducir oraciones en español a glosas de la LSC. Esta función toma una oración en español, la convierte en una secuencia tokenizada, aplica el modelo para predecir la secuencia correspondiente en glosa, y finalmente convierte esa secuencia de nuevo a texto.

Análisis de resultados

En el desarrollo de este estudio, se aplicó una serie de técnicas avanzadas en el campo del procesamiento de lenguaje natural (NLP) con el objetivo de crear un sistema eficaz para traducir el español a la glosa de la lengua de señas colombiana (LSC). Estas técnicas incluyen la tokenización, el análisis morfosintáctico y semántico, y la implementación de redes neuronales recurrentes y algoritmos de aprendizaje automático. A pesar de los esfuerzos significativos y los avances logrados, se encontraron limitaciones sustanciales en la capacidad del sistema para realizar traducciones efectivas, particularmente cuando se enfrentaba a oraciones que no estaban representadas en el corpus de datos utilizado para el entrenamiento de la red neuronal.

Una de las observaciones claves de la investigación se basa en que, mientras la red neuronal mostraba competencia en la traducción de textos incluidos dentro del corpus de entrenamiento, su capacidad para generalizar y aplicar su aprendizaje a oraciones nuevas, similares o completamente diferentes era limitada. Esta limitación se manifiesta en la dependencia de la red neuronal de los datos con los que fue entrenada. Aunque las redes neuronales son capaces de aprender patrones y aplicar estos aprendizajes a nuevos conjuntos de

datos, su efectividad está condicionada por la diversidad y amplitud del corpus de entrenamiento. En este estudio, se evidenció que la red era eficiente al traducir estructuras y frases previamente "vistas", pero su rendimiento disminuye notablemente con oraciones que se desviaban de su entrenamiento.

Por tal motivo, se optó por seguir buscando otras alternativas que permitieran cumplir con los objetivos propuestos, pero sin dejar de lado los avances de la investigación realizados hasta el momento. Es por ello, que se abordará otro tipo de arquitectura basada en Transformers, pues si bien las RNN y las LSTM son opciones preferidas para las tareas secuenciales, los transformers han surgido como un enfoque poderoso para las tareas de NLP, con varias ventajas sobre los modelos tradicionales como RNN y CNN (Nova, 2023).

Tecnologías para la implementación de un avatar intérprete de lengua de señas colombiana a partir de texto en español

El proceso de interpretación de la lengua de señas colombiana es un desafío grande para la inclusión de las personas sordas en Colombia. A medida que se avanza en la era digital, la tecnología se ha convertido en un aliado fundamental para superar las barreras lingüísticas y garantizar que esta parte de la comunidad pueda acceder a la información y comunicarse de manera eficiente y autónoma.

Para lograr este objetivo, es esencial contar con una amplia variedad de herramientas de programación que faciliten el proceso de interpretación de la lengua de señas colombiana y todo el proceso que conlleva. Estas herramientas no solo deben ser eficientes en el análisis del tratamiento del lenguaje natural y la interpretación adecuada de las características de la lengua de señas, sino que también deben ofrecer beneficios significativos en términos de costos de implementación. La facilidad de uso es un factor crucial, ya que cualquier persona en un momento de emergencia debe poder emplear la herramienta final de manera intuitiva y efectiva.

En el contexto de la investigación, es necesario llevar a cabo un estudio de las diferentes herramientas de programación disponibles, evaluando sus características técnicas, su capacidad para procesar información y traducir palabras a la glosa de la lengua de señas colombiana. Además, es esencial analizar los beneficios que ofrecen en términos de accesibilidad y comunicación inclusiva, lo que puede contribuir significativamente a mejorar la calidad de vida de las personas sordas en Colombia.

Durante el proceso de investigación y desarrollo, se encontraron diversas herramientas que al integrarlas contribuyen al proceso de interpretación, sin embargo, presentaban algún tipo de problema, ya sea de compatibilidad, rendimiento o por la complejidad de uso, por tal motivo se generó varias etapas de investigación y desarrollo, en las cuales surgieron nuevas ideas y mejoras al desarrollo.

Primera etapa

El inicio de la investigación de las herramientas de programación se centra en un solo objetivo, el cual fue, permitir que la mayor parte de la población tenga acceso al prototipo final

de avatar traductor, para ello se inició a buscar cuáles son las herramientas tecnológicas más accesibles a la población en general, para lo cual se encontró que las tecnologías móviles logran cubrir un alto porcentaje de la población.

En Colombia se cuenta con altas posibilidades de trabajar con tecnología móvil en dispositivos Android, ya que según estudios de (Ideasti, 2019), en este país cerca del 81.9% de habitantes manejan dispositivos móviles con sistema operativo Android y solamente el 11% manejan dispositivos iOS, por lo cual la aplicación tendría un mayor alcance de usabilidad, generando oportunidades de fácil acceso a la población con discapacidad auditiva y una atención oportuna por parte de cualquier persona en emergencias.

Siguiendo con el transcurso de la primera etapa, ahora el punto importante era buscar un entorno de programación que permitiera desarrollar aplicaciones móviles de manera gratuita, para lo cual se optó por usar Android Studio, herramienta que permite generar desarrollos para aplicaciones en dispositivos con sistema operativo Android, el cual permite realizar los desarrollos de programación en dos lenguajes, el primero es Java, el segundo es Kotlin, dos lenguajes de alto tipado con bastantes características propias que aportan significativamente a la creación del prototipo.

Aplicaciones para el desarrollo front-end

Para el desarrollo del prototipo se optó por la búsqueda de herramientas que facilitaran el desarrollo, donde el conocimiento de los investigadores sería un factor a tener en cuenta, ya que la idea optimizar el tiempo de desarrollo generando oportunidades para avanzar con el diseño del prototipo final.

Java. En el proceso de desarrollo del prototipo, se tomó la decisión inicial de utilizar el lenguaje de programación Java como la primera opción. Esta elección se basó en el conocimiento intelectual de los miembros del equipo de investigación, ya que Java es un lenguaje con el que ya se han realizado proyecto con anterioridad lo que prometía una mayor facilidad de uso y una curva de aprendizaje más suave.

Sin embargo, a medida que se avanzó en el desarrollo del proyecto y se comenzó a diseñar la interfaz de usuario final en Android Studio, se encontró una limitación significativa

en el uso de Java, especialmente en dispositivos móviles. La razón detrás de esta limitación radicó en la necesidad de incorporar gráficos en 3D en la interfaz del prototipo. Java demostró no ser la elección adecuada para este propósito, ya que la complejidad de integrar animaciones en 3D resultó ser un desafío insuperable.

Esta dificultad en la integración de animaciones en 3D se convirtió en un obstáculo significativo en el desarrollo del prototipo, lo que llevó a replantear la elección del lenguaje de programación y explorar otras opciones que pudieran satisfacer los requisitos técnicos de manera más efectiva.

Kotlin. Con el objetivo de superar la limitación en la integración de animaciones en 3D, se inició una investigación sobre el lenguaje de programación más adecuado para la parte que compete al usuario final, es decir, el desarrollo Front-End del proyecto. Durante este proceso, se descubrió que Java, a pesar de ser un lenguaje ampliamente utilizado, presentaba desafíos en términos de integración de animaciones en 3D en la interfaz del prototipo.

Se identificó que Kotlin en la actualidad es especialmente utilizado en el desarrollo de aplicaciones móviles, donde ofrece ventajas significativas ya que este lenguaje destaca por su capacidad para optimizar los recursos de programación y simplificar la creación de funcionalidades de manera concisa en comparación con Java. Estas características fueron esenciales para garantizar un consumo eficiente de recursos en la aplicación (Android Developers, 2017).

Una vez se tomó la decisión de utilizar este nuevo lenguaje, fue necesario indagar sobre el uso del mismo debido a que el conocimiento sobre este era nulo. Posteriormente se logró un proceso efectivo en la integración de gráficos 3D en la interfaz del prototipo. Esto permitió la importación y reproducción automática de las animaciones de manera fluida y sin restricciones, superando así la limitación inicial y mejorando significativamente la calidad y la funcionalidad del prototipo.

Aplicaciones para el desarrollo back-end

En la búsqueda de una herramienta adecuada que permitiera aprovechar las funcionalidades del procesamiento del lenguaje y cumplir con los requisitos de desarrollo, se

tomó la decisión de utilizar Python como el lenguaje de programación principal. Esta elección se basó en una serie de razones fundamentales que hacen de Python una opción altamente apropiada para este proyecto.

Python. En primer lugar, Python se destaca por su naturaleza de lenguaje de programación de bajo tipado, lo que significa que no es necesario declarar explícitamente el tipo de datos de las variables. Esta característica simplifica el proceso de desarrollo, ya que permite libertad de uso por parte del desarrollador, lo cual resulta valioso al trabajar en el procesamiento del lenguaje natural y la traducción de la lengua de señas colombiana.

Otra razón importante para elegir Python es su curva de aprendizaje accesible. Python es conocido por ser un lenguaje amigable para principiantes, por lo tanto, permite que personas con poco conocimiento técnico puedan utilizarlo de manera efectiva. Esto es esencial, ya que facilita la colaboración de personas con diferentes habilidades y experiencias en el proyecto, lo que puede enriquecer el proceso de desarrollo, además, Python es una elección acertada debido a su comunidad activa y su amplia disponibilidad de recursos y bibliotecas. Python es de código abierto, lo que significa que su acceso es gratuito, y cuenta con una abundancia de bibliotecas externas que se pueden integrar en el código. Esto brinda una gran flexibilidad y potencial para enriquecer el algoritmo de traducción con funcionalidades adicionales y herramientas externas que pueden mejorar significativamente el rendimiento y la precisión del sistema (KeepCoding, 2024).

*Arquitectura de software*⁵

Una vez definidas las herramientas de programación para realizar el desarrollo, se inicia a pensar en el funcionamiento, para lo cual se tienen tres diferentes arquitecturas para el desarrollo del software, la primera esa Monolíticas, la siguiente, cliente servidor, y por última, tecnología por microservicios. Cada una de estas arquitecturas que poseen propiedades únicas, se comenzaron a analizar teniendo en cuenta los objetivos de funcionamiento del prototipo; debido a que, el punto de realizar la aplicación en un dispositivo Android es reducir los factores

⁵ Estructura organizativa y el diseño general de un sistema de software, que define cómo sus componentes interactúan y se relacionan entre sí para lograr los objetivos de la aplicación.

de almacenamiento, usabilidad y accesibilidad sin que el usuario final tenga que disponer de muchas características físicas en su dispositivo. A continuación, se inician a analizar cada una de las arquitecturas:

Monolítica. La arquitectura monolítica, que se basa en la ejecución de todo el proceso en el mismo dispositivo, fue analizada en el contexto de este proyecto. Sin embargo, se determinó que esta arquitectura no era la opción más adecuada debido a varias razones fundamentales.

En primer lugar, la arquitectura monolítica tiende a resultar en aplicaciones de mayor tamaño y consumo de recursos, ya que todas las funcionalidades y componentes se encuentran dentro de una sola unidad. Dado que uno de los objetivos clave de este proyecto es reducir los factores de almacenamiento y de uso de recursos en dispositivos móviles, una aplicación monolítica no encajaba con esta premisa (Wagner, 2023).

Además, la usabilidad y accesibilidad son dos aspectos críticos en la aplicación que busca servir a la comunidad sorda. La arquitectura monolítica, al concentrar todo en un solo dispositivo, podría presentar limitaciones en términos de escalabilidad y capacidad para adaptarse a diferentes plataformas y dispositivos. Esto podría resultar en una experiencia de usuario menos flexible y accesible.

Por último, la arquitectura monolítica podría no ser la opción más eficiente en términos de mantenimiento y actualizaciones, ya que cambios en una parte de la aplicación podrían afectar otras áreas, lo que resultaría en un proceso de desarrollo más complejo y costoso.

Cliente servidor. Para el desarrollo de este proyecto, se consideró como principal opción, centrar el funcionamiento en una arquitectura basada en cliente-Servidor, por varias razones, la primera y la más importantes es que en esta arquitectura se tiene un servidor centralizado, el cual realiza los procesos pesados y por medio de peticiones, los dispositivos clientes se conectan y hacen uso de los recursos almacenados en el servidor, la siguiente razón se pensó buscando optimizar el tiempo y el costo en mantenimientos de funcionamiento, ya que los cambios se realizan en un solo dispositivo, sin necesidad de realizar los cambios en cada uno de los dispositivos finales.

Además, la arquitectura cliente-servidor facilita la escalabilidad y la accesibilidad, ya

que el servidor centralizado puede procesar solicitudes de múltiples clientes de manera eficiente. Esto garantiza que la aplicación pueda funcionar de manera efectiva en diferentes dispositivos Android y proporcionar una experiencia de usuario consistente y accesible (Blancarte, 2014).

Microservicios. La arquitectura de microservicios, aunque es valiosa en muchos casos, no se considera la opción más apropiada para este proyecto. Esto se debe a que la naturaleza de los microservicios implica la división de una aplicación en múltiples componentes independientes, cada uno con su propia funcionalidad específica. Aunque es un recurso muy eficiente, puede ser beneficiosa en proyectos de gran escala o con necesidades extremas de escalabilidad, en el contexto de proporcionar una aplicación móvil eficiente y accesible para la interpretación de la lengua de señas colombiana en dispositivos Android, esta división podría generar una complejidad innecesaria. Además, los microservicios suelen requerir una infraestructura de red más elaborada para facilitar la comunicación entre estos componentes separados, lo que podría aumentar la latencia ⁶y el consumo de datos en dispositivos móviles, algo que es crucial evitar en este proyecto. Además, la gestión y el mantenimiento de múltiples microservicios pueden resultar más complejos en comparación con un enfoque más monolítico, lo que podría traducirse en costos y cargas de trabajo adicionales (Harris, 2022).

Una vez que se seleccionó Python como la herramienta para desarrollar el código Backend⁷, Android Studio para el Frontend⁸ y se determinó que la arquitectura adecuada para la implementación del proyecto sería Cliente-Servidor, el siguiente paso en la primera etapa de investigación y desarrollo consistió en establecer la comunicación entre las dos partes involucradas.

⁶ Tiempo que tarda un sistema o dispositivo en responder a una solicitud o en procesar una acción

⁷ Parte de un sistema de software que se encarga de gestionar los procesos y la lógica de funcionamiento en el servidor.

⁸ Parte de un sistema de software que se encarga de la interfaz de usuario y la presentación de la información.

Comunicación cliente-servidor

Para lograr esto, se empleó el uso de una API REST⁹, mediante la cual se envían frases desde el dispositivo Android al servidor y se reciben las respuestas correspondientes. Esta comunicación se llevó a cabo a través de solicitudes HTTP¹⁰ utilizando los métodos GET y SET.

Además, se implementó un protocolo basado en TCP/IP¹¹ para establecer la dirección IP a la que se realizarán las peticiones y el puerto lógico al cual está sujeto el servidor. También se configuraron restricciones de acceso para determinar qué direcciones IP podrían acceder al servidor, lo que contribuyó a garantizar la seguridad y la integridad de la comunicación (IONOS, 2023).

El proceso para la primera etapa culminó en el punto anteriormente mencionado, ya que por el momento no se tenía el conocimiento adecuado para seguir con el desarrollo, aunque fue un proceso largo y con varias dificultades. Fue un primer acercamiento al funcionamiento deseado, donde se tenía ya establecida sólidamente la manera en la cual iba a funcionar el prototipo de avatar intérprete, dando paso a la siguiente etapa de investigación.

Segunda etapa

En esta etapa de la investigación, el objetivo principal se centra en explorar a fondo los avances en el campo de la inteligencia artificial¹². Esto se debe a que, llevar a cabo un proceso de traducción de la lengua de señas colombiana es una tarea compleja que requiere un sólido conocimiento y comprensión de las herramientas y técnicas disponibles en este campo. La inteligencia artificial, con su capacidad para el procesamiento de lenguaje natural se presenta como un componente esencial en la creación de soluciones efectivas para la traducción de lengua de señas.

⁹ Conjunto de reglas y convenciones que permiten a las aplicaciones web comunicarse y compartir datos de manera eficiente al utilizar los métodos HTTP.

¹⁰ Mensajes enviados por un cliente a un servidor web para solicitar recursos.

¹¹ Conjunto de protocolos de comunicación utilizados para conectar computadoras y dispositivos en redes de computadoras y permitir la transferencia de datos a través de Internet.

¹² Creación de programas y sistemas informáticos que pueden simular procesos de pensamiento humano, como el razonamiento, la resolución de problemas y la comprensión del lenguaje natural.

Durante este periodo de la investigación, se buscó conocer a fondo las últimas tendencias, algoritmos y herramientas relacionadas con la inteligencia artificial aplicada a la traducción de idiomas, ya que directamente no se tienen muchos recursos en la traducción de lengua de señas y mucho menos en la LSC, donde la idea de esta investigación es entender cómo funciona el proceso para en una posterior etapa intentar crear un modelo que logre producir la glosa de la lengua de señas a partir de una frase en español.

En esta fase de la investigación, la falta de conocimientos sobre la lengua de señas colombiana comenzó a convertirse en un desafío, por lo que se emprendió la tarea de explorar el tema. A través del uso del diccionario de lengua de señas colombiana se obtuvieron ejemplos de oraciones con su respectiva glosa. También se interactuó con individuos que tienen un profundo entendimiento de esta lengua y que forman parte de la comunidad sorda, para ser exactos colaboraron dos personas, una docente de la Universidad de Boyacá, quien es intérprete de la LSC y un joven que hace parte de la comunidad sorda. Este enfoque permitió adentrarse en los fundamentos necesarios para desarrollar un modelo efectivo de predicción.

En el marco de la investigación, se comienza un proceso para determinar las señas más utilizadas en situaciones de emergencia. Para ello, se recurrió a un video disponible en la plataforma de acceso público YouTube, en el cual miembros del cuerpo de bomberos de otro país representaban las señas relevantes. Una vez recopiladas estas frases, se trabajó en conjunto con la comunidad sorda e intérpretes de la lengua de señas para realizar una transcripción escrita precisa de estas señas, lo que se documentó en un archivo con formato JSON como se evidencia en el Anexo B.

En esta etapa, se desarrolló un fragmento de código que desempeñó un papel crucial. Este código se encargó de analizar las frases presentes en el archivo y mediante un proceso de comparación de similitud, determinó cuál de las frases disponibles era la más cercana a la que se ingresaba. Para llevar a cabo esta comparación de similitud, se implementó un modelo basado en Transformers, que es una tecnología de Inteligencia Artificial que permite analizar y comprender el significado de las oraciones (Santander, 2021).

En la fase de implementación, se adaptó el sistema para que los usuarios desde sus dispositivos Android pudieran escribir una frase y enviarla al servidor. El servidor, a su vez, verifica esta frase con la base de datos existente y utilizando el modelo Sentence-Transformer basado en Transformers para analizar la similitud. Luego, se envía de vuelta al dispositivo

Android el porcentaje de similitud y la glosa con la cual se realizó la similitud, permitiendo que la aplicación móvil en una futura etapa visualice la animación de la seña correspondiente.

En esta etapa se presentaron problemas por la complejidad del tema, pero se logró comprender varios aspectos importantes para la siguiente etapa del proyecto.

Tercera etapa

En búsqueda de encontrar algún proceso que facilitara la traducción del español a la glosa de lengua de señas se inició a identificar redes neuronales con varios modelos de entrenamiento, para lo cual después de una búsqueda e implementación del modelo de red neuronal de tipo LSTM, la cual por sus atributos y técnicas para el tratamiento del lenguaje natural parecía la mejor opción, pero al iniciar a ver los recursos tan limitados que se tenían para iniciar con un entrenamiento desde cero, se decidió buscar otra alternativa empleando alguna de las ramas de la inteligencia artificial para lo cual se encontraron los modelos transformers, sin dejar de lado el proceso de investigación previamente mencionado en el capítulo 1. Los transformers emplean una técnica donde el modelo va aprendiendo automáticamente con los datos con los que se realiza el entrenamiento, sin embargo, iniciaron nuevamente las limitaciones, puesto que el material de conocimiento era limitado, no se tenía ningún conocimiento acerca de la glosa de lengua de señas colombiana, para lo cual fue necesario realizar un conjunto de datos que ayudara a efectuar un entrenamiento, por medio de frases encontradas y disponibles en el diccionario de lengua de señas colombiana. Se encontraron frases que ejemplificaban situaciones de la vida real, se armó un conjunto de aproximadamente 230 frases, con las cuales se inició el proceso de entrenamiento, donde la estructura del archivo consistía en dos partes, la primera era la representación textual de la glosa y la segunda su traducción en español, esto con el fin de tener un diccionario propio, el cual sería la base para realizar el proceso de traducción automática.

Una vez se tenía la información, era necesario saber qué tipo de modelo de entrenamiento se iba a utilizar, y se tomó la decisión de emplear los Transformers que es una arquitectura de red neuronal muy adecuada para el procesamiento del lenguaje natural NLP, además, tiene mucha popularidad en procesos actuales, por tal motivo y por su alcance se inició a buscar un lenguaje de programación que se ajustara a la necesidad de desarrollo para lo cual

con ayuda de Python desde la plataforma de Google Colab se inició con la adaptación de un modelo de traducción de idiomas encontrado en GitHub cuyo nombre es “Traducción de texto con Redes Transformer” del autor Miguel Sotaquirá.

Fue necesario iniciar a estudiar la red neuronal de tipo Transformer encontrado, para saber cómo funcionaba y de qué manera se buscaría realizar una adaptación a la situación requerida, para eso fue necesario entender línea a línea el código fuente del modelo encontrado.

Para iniciar el proceso de desarrollo lo primero fue comprender la forma en la cual se accedería a la información, para ello se identificó que se accede a las columnas del conjunto de datos, donde se separan las mismas y se crea una matriz donde se almacenan los datos para ser procesados en una posterior etapa. A continuación, se procede con el inicio del entrenamiento del modelo, para lo cual se accede a los datos almacenados en la matriz, donde cada uno de los datos del conjunto debía ser separado, proceso que se llama tokenizar, que consiste en dividir cadenas de caracteres en partes más pequeñas, el siguiente paso consistió en separar los token, ya que se debía reconocer cuál era el inicio de una frase y cuál era su final para lo cual se usaron tres etiquetas, que identificaban el principio y el final de cada frase.

Figura 1

Etiquetas de tokenización

```
def build_token_dict(token
token_dict = {
    '<PAD>': 0,
    '<START>': 1,
    '<END>': 2
}
```

Fuente: autor de la investigación

Como se puede observar en la figura, las etiquetas tienen una función en específico, ya que es importante conocer el inicio y el final de cada oración, pero además al estar trabajando con una matriz, se requería que todos los datos tuvieran una misma longitud, para lo cual se usa la etiqueta <PAD> que rellenaba la oración con el número 0 hasta cumplir con la longitud deseada y se puedan procesar por lotes durante el proceso de entrenamiento.

El proceso ya está listo a esta altura de la preparación de la información para el entrenamiento, pero sin embargo, se emplea una nueva forma de acceder a la matriz, ya que es muy tedioso acceder a la información textualmente, el proceso es muy largo y la optimización de los recursos se ve afectada, para lo cual se creó una matriz con índices, que contenía los mismos datos pero esta vez representados con las posiciones de la matriz y con esto los datos estaban listos para iniciar con el diseño de la red transformer. Para realizar este proceso, se hace uso de una función de otro transformer “Keras_ Transformer” para la construcción del modelo Transformer con parámetros directamente especificados, incluyendo la función de pérdida de entropía cruzada¹³ y el optimizador adam¹⁴. Para realizar este proceso en la adaptación propia, fue necesario realizar un entrenamiento basado en prueba y error, ya que fue necesario manualmente verificar cada uno de los parámetros que más se ajustaran al modelo en entrenamiento.

Para lo anterior, se documentó cada prueba, donde se observaron los resultados obtenidos con diferentes parámetros, y se decidió dejar los parámetros observados en la figura 2, ya que se ajustaron al comportamiento deseado.

Las demás pruebas realizadas están disponibles en el Anexo C donde se evidencian los cambios realizados en los parámetros de la creación de la red Transformer y en el entrenamiento de este.

¹³ Función empleada para problemas de clasificación que permite realizar predicciones más precisas en el entrenamiento de una red neuronal.

¹⁴(Adaptive Moment Estimation) es un algoritmo de optimización que se utiliza para ajustar los pesos del modelo durante el entrenamiento.

Figura 2*Creación de la red transformer*

```
[ ] # Crear la red transformer
    model = get_model(
        token_num = max(len(source_token_dict), len(target_token_dict)),
        embed_dim = 32,
        encoder_num = 2,
        decoder_num = 2,
        head_num = 4,
        hidden_dim = 128,
        dropout_rate = 0.05,
        use_same_embed = False,
    )
    model.compile('adam', 'sparse_categorical_crossentropy')
    model.summary()
```

Fuente: autor de la investigación

El siguiente paso en la adaptación de la red neuronal, consistió en realizar el entrenamiento del modelo anteriormente analizado, donde se empleó un entrenamiento con 800 epochs¹⁵, por último antes de estar listo el modelo para realizar una traducción de español a la glosa de lengua de señas, fue necesario eliminar las etiquetas de inicio y fin, para poder enviar al usuario final la información de manera agradable a la vista y que se entendiera la traducción sin necesidad de una capacitación previa.

Pruebas de desarrollo del modelo.

El primer paso fue la creación del material de conocimiento, el cual se realizó en una hoja de Excel.

¹⁵ Cantidad de veces en las cuales los datos de entrenamiento pasan a través de todo el conjunto de datos de entrenamiento durante el proceso de entrenamiento de una red neuronal.

Figura 3

Creación de diccionario de glosa

Espanol	Glosa
La primera mujer no era muy bonita.	MUJER PRIMERO BONITO REGULAR
Un hombre que monta en bicicleta mira hacia arriba.	HOMBRE PEDALEAR-BICICLETA VER
Los homosexuales protestaron en el centro de la ciudad.	CENTRO PROTESTA MARCHA HOMOSEXUAL
Un niño monta en una bicicleta.	NIÑO PEDALEAR-BICICLETA
Nosotros dos fuimos elegidos para representar al colegio en el concurso.	COLEGIO ELEGIR PRODUAL REPRESENTAR CONCURSO
Cada persona debe llevar la comida al paseo.	CADA-UNO PERSONA PASEAR COMIDA DEBER LLEVAR
El niño tiene una infección en la boca, su mamá le compró medicamento	NIÑO BOCA INFECCIÓN PROPOS MAMÁ COMPRAR DROGA
La cara siempre debe protegerse del sol.	CARA DEBER SIEMPRE SOL CONTROLAR
El payaso pintó de rojo sus cejas y sus mejillas.	PAYASO CEJA MEJILLA COLOR ROJO PINTAR
En la clase de biología el profesor enseñó las partes del cuerpo.	CLASE BIOLOGÍA PROFESOR ENSEÑAR CUERPO PARTE
El bebé nació con las orejas muy grandes.	BEBÉ NACER OREJA GRANDE
El niño irá al hospital porque le harán una cirugía en el pene.	NIÑO IR HOSPITAL CIRUGÍA PENE
El árbitro no pudo pitar el partido, le dolían las piernas.	ÁRBITRO PIERNA DOLOR NO-PODER PITAR PARTIDO

Fuente: autor de la investigación

Descarga e instalación del transformer necesario para el entrenamiento de la red neuronal.

Figura 4

Instalación de dependencias

```

pip install keras-transformer
... Collecting keras-transformer
  Downloading keras-transformer-0.40.0.tar.gz (9.7 kB)
  Preparing metadata (setup.py) ... done
  Collecting keras-pos-embd==0.13.0 (from keras-transformer)
  Downloading keras-pos-embd-0.13.0.tar.gz (5.6 kB)
  Preparing metadata (setup.py) ... done
  Collecting keras-multi-head==0.29.0 (from keras-transformer)
  Downloading keras-multi-head-0.29.0.tar.gz (13 kB)
  Preparing metadata (setup.py) ... done
  Collecting keras-layer-normalization==0.16.0 (from keras-transformer)
  Downloading keras-layer-normalization-0.16.0.tar.gz (3.9 kB)

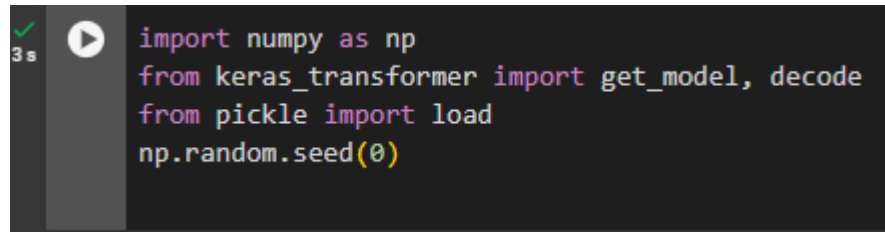
```

Fuente: autor de la investigación

Para continuar con el proceso se importaron las librerías necesarias para el entrenamiento de la red neuronal.

Figura 5

Importación de módulos y funciones



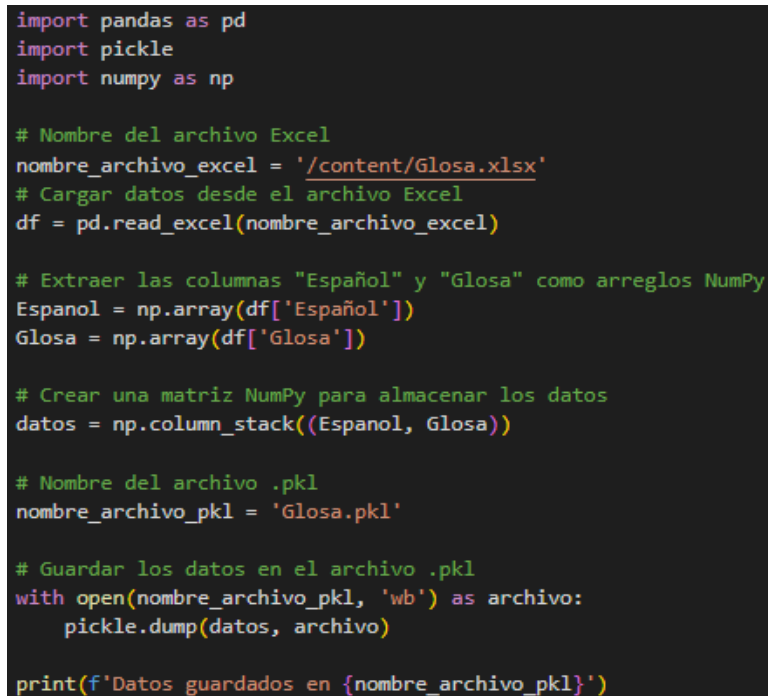
```
import numpy as np
from keras_transformer import get_model, decode
from pickle import load
np.random.seed(0)
```

Fuente: autor de la investigación

Después se realizó el cargue de los datos, y la extracción de la información en los vectores separados y por último se creó la matriz con los datos ya preprocesados.

Figura 6

Preparación de los datos



```
import pandas as pd
import pickle
import numpy as np

# Nombre del archivo Excel
nombre_archivo_excel = '/content/Glosa.xlsx'
# Cargar datos desde el archivo Excel
df = pd.read_excel(nombre_archivo_excel)

# Extraer las columnas "Español" y "Glosa" como arreglos NumPy
Espanol = np.array(df['Español'])
Glosa = np.array(df['Glosa'])

# Crear una matriz NumPy para almacenar los datos
datos = np.column_stack((Espanol, Glosa))

# Nombre del archivo .pkl
nombre_archivo_pkl = 'Glosa.pkl'

# Guardar los datos en el archivo .pkl
with open(nombre_archivo_pkl, 'wb') as archivo:
    pickle.dump(datos, archivo)

print(f'Datos guardados en {nombre_archivo_pkl}')
```

Fuente: autor de la investigación

Una vez se tenía la información almacenada, se procedió a leer los datos creados en la matriz y, para evidenciar que sí se accedía a los recursos, se iban realizando impresiones por

consola ¹⁶, las cuales sirvieron de prueba en cada etapa, ya que se comprobaba si la información estaba correctamente almacenada para ser procesada en etapas posteriores. Lo siguiente a realizar fue la creación de los tokens para cada una de las oraciones almacenadas en la matriz.

Figura 7

Proceso de tokenización

```
0. [6] # Leer set de entrenamiento
      filename = '/content/Glosa.pkl'

      dataset = load(open(filename, 'rb'))
      print(dataset[10,0])
      print(dataset[10,1])

      El bebé nació con las orejas muy grandes.
      BEBÉ NACER OREJA GRANDE

0. [7] # Crear "tokens"
      source_tokens = []
      for sentence in dataset[:,0]:
          source_tokens.append(sentence.split(' '))
      print(source_tokens[10])

      target_tokens = []
      for sentence in dataset[:,1]:
          target_tokens.append(sentence.split(' '))
      print(target_tokens[10])

      ['El', 'bebé', 'nació', 'con', 'las', 'orejas', 'muy', 'grandes.']
      ['BEBÉ', 'NACER', 'OREJA', 'GRANDE']
```

Fuente: autor de la investigación

Para poder diferenciar el inicio y el final de cada frase se optó por la creación de etiquetas, donde oración por oración se realizaba un proceso que diferenciaba.

¹⁶ Según lo menciona Maldonado (2024) “Interfaz o entorno de desarrollo monocolor que todo ordenador posee y que funciona por comandos de texto u órdenes escritas”

Figura 8*Construcción de diccionario de tokens*

```

0s ▶ def build_token_dict(token_list):
    token_dict = {
        '<PAD>': 0,
        '<START>': 1,
        '<END>': 2
    }
    for tokens in token_list:
        for token in tokens:
            if token not in token_dict:
                token_dict[token] = len(token_dict)
    return token_dict

0s [9] source_token_dict = build_token_dict(source_tokens)
    target_token_dict = build_token_dict(target_tokens)
    target_token_dict_inv = {v:k for k,v in target_token_dict.items()}

    print(source_token_dict)
    print(target_token_dict)
    print(target_token_dict_inv)

{'<PAD>': 0, '<START>': 1, '<END>': 2, 'La': 3, 'primera': 4, 'mujer': 5, 'no'
{'<PAD>': 0, '<START>': 1, '<END>': 2, 'MUJER': 3, 'PRIMERO': 4, 'BONITO': 5,
{0: '<PAD>', 1: '<START>', 2: '<END>', 3: 'MUJER', 4: 'PRIMERO', 5: 'BONITO',

```

Fuente: autor de la investigación

En el proceso de la asignación de las etiquetas, se creó una etiqueta que apoyaba el proceso, la cual rellenaba los campos restantes de la matriz para que cada oración tuviera la misma dimensión.

Figura 9*Preprocesamiento de los datos*

```

['<START>', 'El', 'bebé', 'nació', 'con', 'las', 'orejas', 'muy', 'grandes.', '<END>', '<PAD>', '<PAD>', '<PAD>', '<PAD>', '<PAD>', '<PAD>',

```

Fuente: autor de la investigación

Observando que el proceso se realizaba por medio de las palabras, y que una manera óptima de realizar el proceso era por medio de índices numéricos, se creó una matriz numérica

con la posición de cada palabra.

Figura 10

Mapeo de tokens a sus respectivos índices

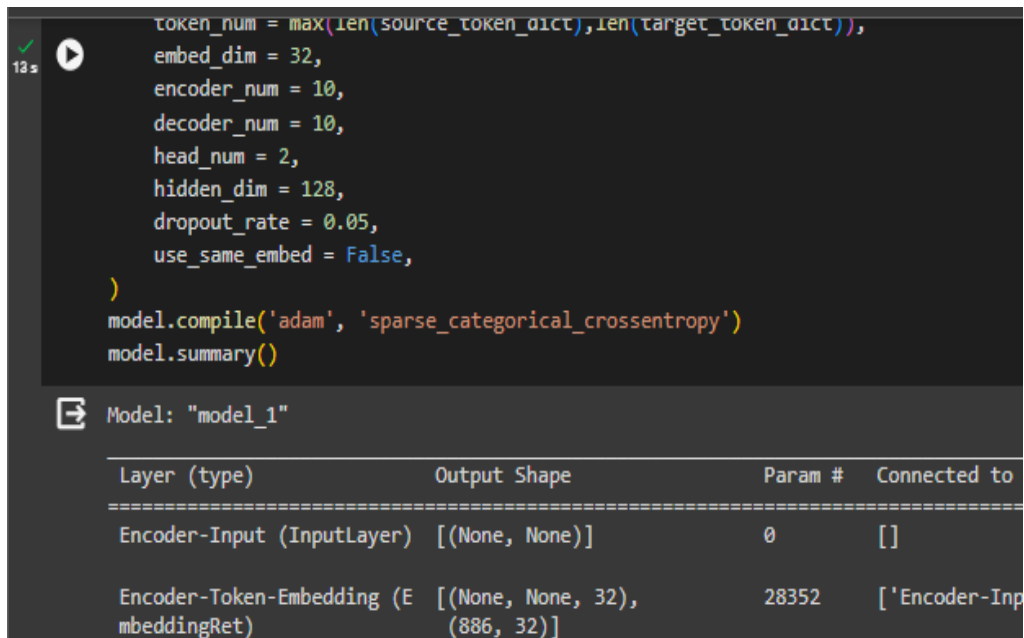
```
[1, 45, 74, 75, 76, 71, 77, 8, 78, 2, 0, 0, 0, 0, 0, 0, 0]
```

Fuente: autor de la investigación

Para la creación del modelo, los parámetros se ajustaron manualmente con el fin de determinar cuáles eran los valores que se ajustaban al requerimiento del entrenamiento.

Figura 11

Creación y ajuste de red neuronal



```

token_num = max(len(source_token_dict), len(target_token_dict)),
embed_dim = 32,
encoder_num = 10,
decoder_num = 10,
head_num = 2,
hidden_dim = 128,
dropout_rate = 0.05,
use_same_embed = False,
)
model.compile('adam', 'sparse_categorical_crossentropy')
model.summary()

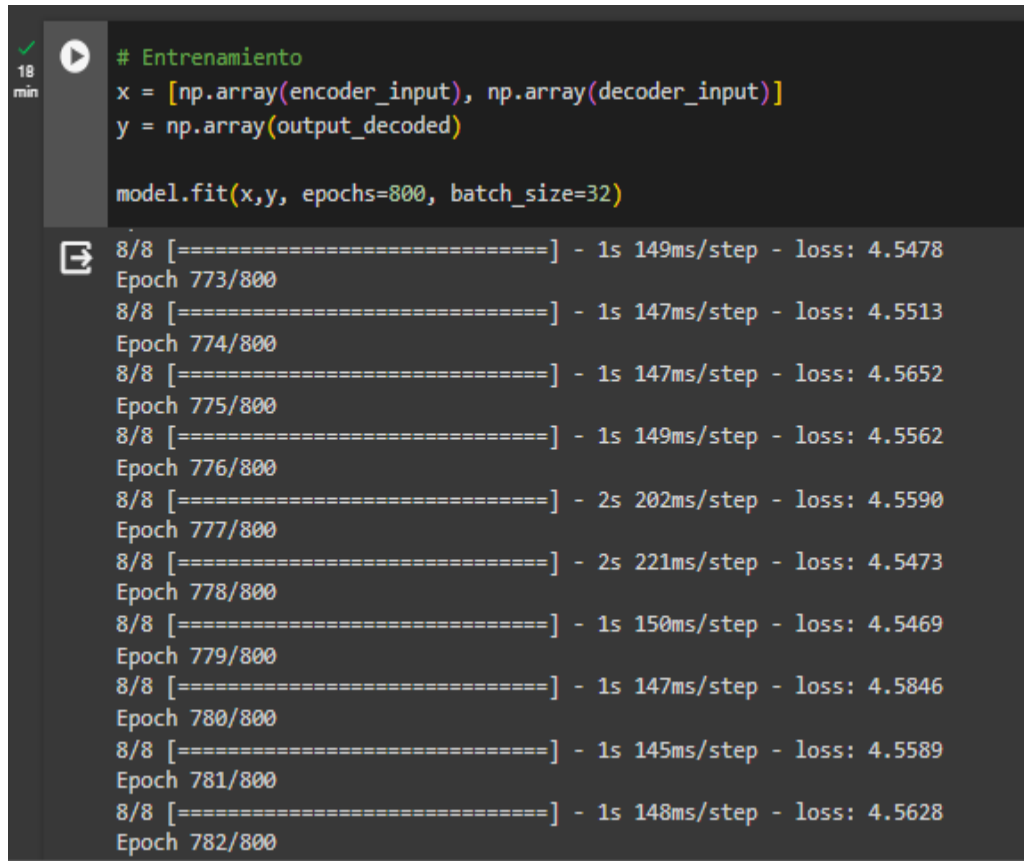
```

Model: "model_1"

Layer (type)	Output Shape	Param #	Connected to
Encoder-Input (InputLayer)	[(None, None)]	0	[]
Encoder-Token-Embedding (Embedding)	[(None, None, 32), (886, 32)]	28352	['Encoder-Input']

Fuente: autor de la investigación

Una vez creada la red, se procedió a realizar el entrenamiento, para lo cual se emplearon 800 etapas. Durante cada una de las etapas se recorría todo el modelo con la información de entrenamiento.

Figura 12*Entrenamiento de la red neuronal*


```

# Entrenamiento
x = [np.array(encoder_input), np.array(decoder_input)]
y = np.array(output_decoded)

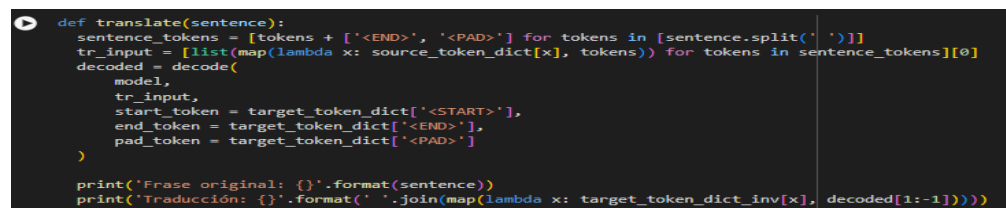
model.fit(x,y, epochs=800, batch_size=32)

8/8 [=====] - 1s 149ms/step - loss: 4.5478
Epoch 773/800
8/8 [=====] - 1s 147ms/step - loss: 4.5513
Epoch 774/800
8/8 [=====] - 1s 147ms/step - loss: 4.5652
Epoch 775/800
8/8 [=====] - 1s 149ms/step - loss: 4.5562
Epoch 776/800
8/8 [=====] - 2s 202ms/step - loss: 4.5590
Epoch 777/800
8/8 [=====] - 2s 221ms/step - loss: 4.5473
Epoch 778/800
8/8 [=====] - 1s 150ms/step - loss: 4.5469
Epoch 779/800
8/8 [=====] - 1s 147ms/step - loss: 4.5846
Epoch 780/800
8/8 [=====] - 1s 145ms/step - loss: 4.5589
Epoch 781/800
8/8 [=====] - 1s 148ms/step - loss: 4.5628
Epoch 782/800

```

Fuente: autor de la investigación

El siguiente paso consistió en preparar la información para la salida de los datos del modelo entrenado, para lo cual se requirió eliminar las etiquetas.

Figura 13*Procesamiento de los datos*


```

def translate(sentence):
    sentence_tokens = [tokens + ['<END>', '<PAD>'] for tokens in [sentence.split(' ')]]
    tr_input = [list(map(lambda x: source_token_dict[x], tokens)) for tokens in sentence_tokens][0]
    decoded = decode(
        model,
        tr_input,
        start_token = target_token_dict['<START>'],
        end_token = target_token_dict['<END>'],
        pad_token = target_token_dict['<PAD>']
    )

    print('Frase original: {}'.format(sentence))
    print('Traducción: {}'.format(' '.join(map(lambda x: target_token_dict_inv[x], decoded[1:-1]))))

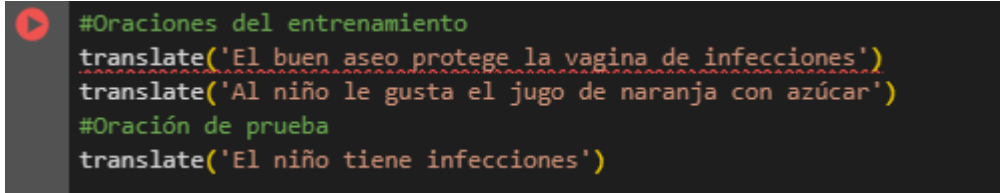
```

Fuente: autor de la investigación

Por último, ya está listo el modelo para la traducción.

Figura 14

Resultados del entrenamiento



```
#Oraciones del entrenamiento
translate('El buen aseo protege la vagina de infecciones')
translate('Al niño le gusta el jugo de naranja con azúcar')
#Oración de prueba
translate('El niño tiene infecciones')
```

Fuente: autor de la investigación

Este modelo se entrenó correctamente. Lastimosamente existieron limitaciones en el acceso de la información por tal motivo la exactitud con la que se predecía la traducción no fue la ideal, ya que se evidenció que modelos que realizaban procesos de traducción de idiomas, eran entrenados con miles de datos y el modelo implementado fue entrenado con una base de datos de aproximadamente 230 datos, por tal motivo el modelo no se logró implementar en el prototipo final del avatar intérprete de lengua de señas.

Selección y utilización de la herramienta más adecuada para la creación del prototipo de avatar para señas de atención de emergencia en LSC con expresiones faciales y manuales

En la búsqueda por dotar al prototipo de avatar 3D con la capacidad de representar las señas de atención de emergencia de la LSC, se llevó a cabo un cuidadoso proceso de selección de herramientas. En este capítulo se aborda el procedimiento sobre cómo se eligió la aplicación indicada para el desarrollo del avatar, desde la fase inicial de exploración hasta la decisión definitiva.

Análisis de herramientas

El primer paso consistió en analizar las herramientas disponibles para la creación y animación de avatares. Diversas opciones se presentaron en el panorama tecnológico, entre las cuales se destacaron nombres como DAZ Studio, MakeHuman, Autodesk Maya, VRoid Studio, Blender, y otras que han demostrado su eficiencia en proyectos similares.

Tabla 1

Herramientas de diseño 3D

Herramienta	Ventaja	Desventaja
DAZ Studio	<ul style="list-style-type: none"> ● Especializado en modelado y animación de personajes humanos. ● Interfaz intuitiva y fácil de usar. ● Gran biblioteca de activos y modelos pre-diseñados. ● Gratuito para usuarios básicos. 	<ul style="list-style-type: none"> ● Menos versátil para modelado general en comparación con otras herramientas. ● Puede requerir compras adicionales para acceder a ciertos activos y funciones.

MakeHuman	<ul style="list-style-type: none"> ● Foco en la creación rápida y sencilla de modelos humanos realistas. ● Completamente gratuito y de código abierto. ● Interfaz sencilla y fácil de aprender. ● Compatibilidad con otras herramientas de modelado 3D. 	<ul style="list-style-type: none"> ● Menos funcionalidades en comparación con herramientas más avanzadas. ● Limitado principalmente a la creación de humanos.
Autodesk Maya	<ul style="list-style-type: none"> ● Ampliamente utilizado en la industria para animación, modelado y efectos visuales. ● Potente conjunto de herramientas para modelado, animación y renderizado. ● Alta personalización y extensión con scripts y plugins. 	<ul style="list-style-type: none"> ● Curva de aprendizaje empinada para principiantes. ● Costo elevado de licencia.
VRoid Studio	<ul style="list-style-type: none"> ● Especializado en la creación de avatares estilo anime. ● Interfaz intuitiva y fácil de usar. ● Gratuito. ● Comunidad activa y creciente. 	<ul style="list-style-type: none"> ● Menos versátil para otros estilos que no sean anime. ● Funciones de animación limitadas en comparación con otras herramientas.
Blender	<ul style="list-style-type: none"> ● Gratuito y de código abierto. ● Conjunto de herramientas completo para modelado, animación y renderizado. 	<ul style="list-style-type: none"> ● Curva de aprendizaje inicial puede ser desafiante. ● Puede ser menos eficiente en ciertas tareas en comparación con

	<ul style="list-style-type: none"> ● Comunidad grande y recursos de aprendizaje abundantes. ● Actualizaciones constantes y mejoras. 	software específico de la industria.
Unity	<ul style="list-style-type: none"> ● Ampliamente utilizado para el desarrollo de juegos y experiencias interactivas. ● Buena integración con herramientas de modelado y animación. ● Gran cantidad de recursos y comunidad de apoyo. ● Flexible y adaptable para diferentes plataformas. 	<ul style="list-style-type: none"> ● Más enfocado en el desarrollo de juegos que en la animación pura. ● Puede requerir conocimientos de programación para aprovechar al máximo.
Unreal Engine	<ul style="list-style-type: none"> ● Motor gráfico de alta calidad para juegos y simulaciones. ● Buenas herramientas para animación y creación de entornos. ● Gratuito para empezar, con un modelo de pago basado en ingresos. ● Potente sistema de iluminación y renderizado en tiempo real. 	<ul style="list-style-type: none"> ● Curva de aprendizaje empinada, especialmente para principiantes. ● Requiere un hardware relativamente potente para aprovechar todas sus funciones.

Fuente: autor de la investigación

Elección de Daz Studio 3D

Después de la investigación realizada que se basó en fuentes de información diversas como foros, vídeos en YouTube, documentos técnicos y otros informes, se decidió inicialmente por DAZ Studio. Esta elección se fundamentó en la completa gama de funcionalidades que ofrecía, por lo cual se comenzó el proceso de modelado del avatar en este software.

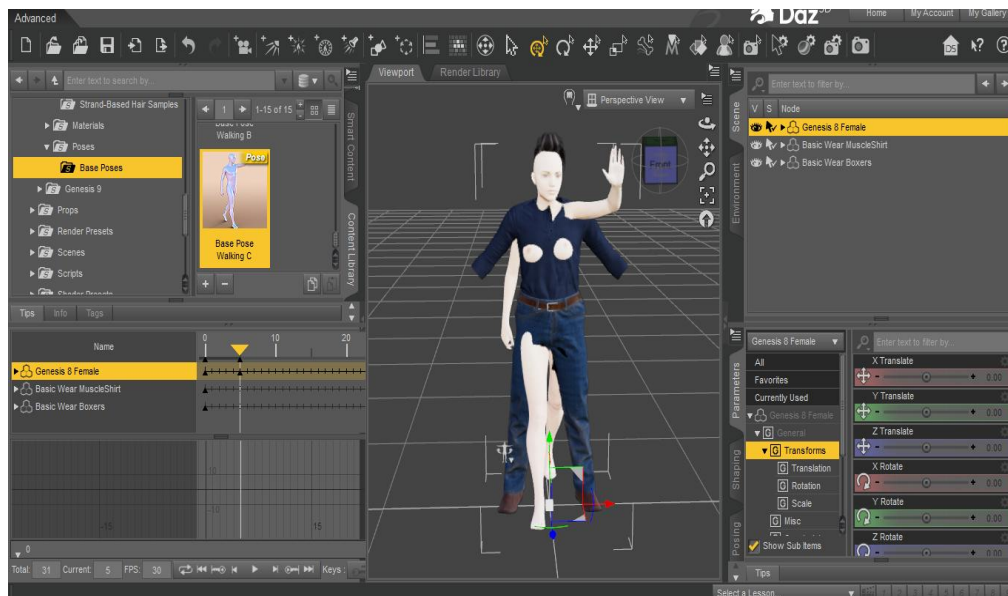
Desafíos de compatibilidad

Después de realizar la primera señal de emergencia e intentar integrarla con el desarrollo de la aplicación móvil, se presentaron desafíos significativos de compatibilidad.

Al momento de exportar el archivo en el formato requerido (GLB) para el funcionamiento de este en la app, se generaban errores en el modelo ocasionando inconvenientes que comprometían la integración efectiva del avatar en el entorno móvil.

Figura 15

Animación en Daz Studio 3D



Fuente: autor de la investigación

Transición a Blender

Ante las dificultades encontradas, se tomó la decisión estratégica de buscar una alternativa que solventara las necesidades específicas del proyecto, por lo cual se realizó nuevamente una búsqueda de la herramienta más adecuada para cumplir el propósito de realizar la animación, con lo cual se eligió Blender, Un software ampliamente reconocido en el ámbito de la animación y modelado 3D, el cual se destacó por cumplir con todos los requisitos necesarios para modelar el avatar destinado a la aplicación móvil y también se tuvo en cuenta la variedad de hardware disponible, ya que su compatibilidad con una amplia gama de dispositivos facilita la accesibilidad y el uso generalizado (Chirivella, 2022).

Incorporación de Mixamo en Blender

Dado el ritmo acelerado de avances tecnológicos, se considera la posibilidad de realizar futuras actualizaciones en las herramientas y complementos seleccionados, como Mixamo en Blender, con el fin de aprovechar nuevas funcionalidades y mejoras que puedan surgir. Esto garantizará que el proceso de animación de avatares siga siendo más fácil y eficiente incluso con los cambios y mejoras en el entorno tecnológico. Mixamo, es un servicio web que permite animar personajes en 3D y usa métodos que permiten automatizar el proceso de animación, lo cual hace que, para personas sin experiencia en creación y movimiento de avatares, el modelado sea más fácil de realizar (Williamson, 2022).

Elaboración de las animaciones

Se elaboraron aproximadamente 12 animaciones que representaban cada palabra de la glosa para las señas de atención de emergencia en la LSC. Estas animaciones fueron cuidadosamente creadas utilizando Blender, aprovechando su potencial y versatilidad para el modelado y animación de personajes 3D.

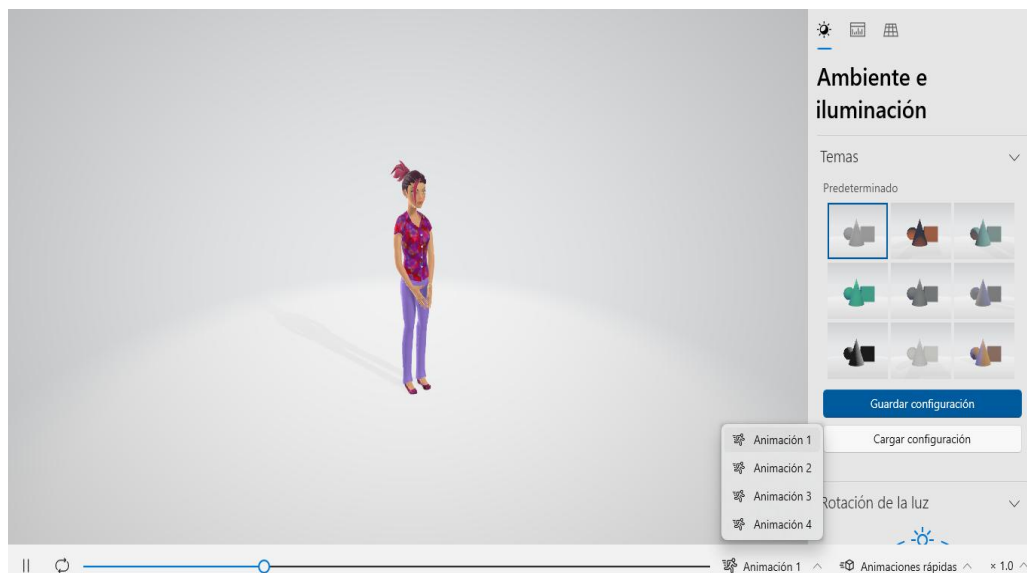
Cada animación capturó con precisión los movimientos manuales requeridos para cada seña, asegurando una representación precisa y fiel de la LSC. El proceso involucró la configuración meticulosa de las articulaciones del avatar y la sincronización de los

movimientos.

Una vez completadas las animaciones individuales, se compilaron y optimizaron en un único archivo en formato GLB (Binary GL Transmission Format). Este formato es ampliamente compatible con aplicaciones web y móviles, lo que permitió su integración en el aplicativo destinado a mostrar las señas de atención de emergencia.

Figura 16

Animación GLB



Fuente: autor de la investigación

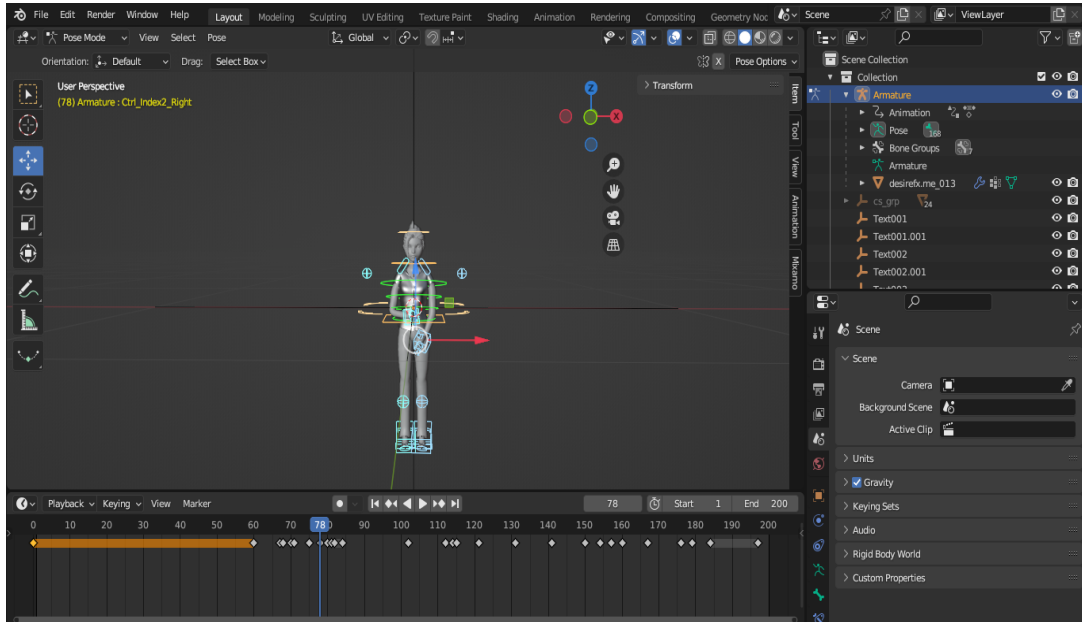
El archivo GLB contiene todas las animaciones de cada palabra de las glosas, permitiendo que la aplicación reconozca y reproduzca cada una en el orden correcto, ofreciendo una experiencia fluida a los usuarios. Esto garantiza que las señas de emergencia se presenten de manera precisa y comprensible, facilitando la comunicación y la respuesta efectiva en situaciones de emergencia.

La decisión de animar cada palabra de la glosa se tomó con el objetivo de reutilizarlas en diferentes contextos y ampliar el alcance del aplicativo móvil en el futuro. Sin embargo, las animaciones de cada palabra de la glosa generan pequeños cortes visuales al cambiar entre las animaciones de cada oración, así como al comenzar a interpretar una oración nueva que se ingrese. Esto se debe a la naturaleza de la animación palabra por palabra, donde cada cambio

de palabra puede crear una interrupción visual.

Figura 17

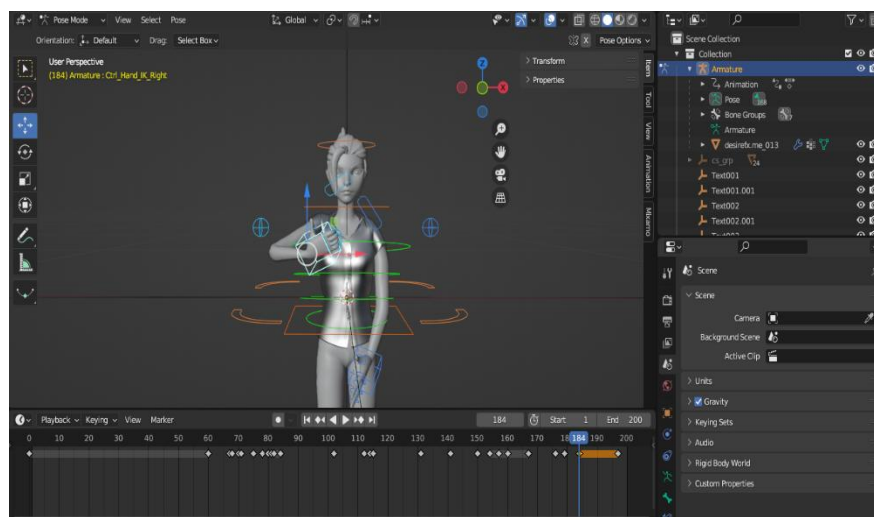
Proceso de animación en Blender



Fuente: autor de la investigación

Figura 18

Animación en Blender de palabra bien



Fuente: autor de la investigación

Integración final del prototipo de avatar para la traducción de frases en español a la glosa de la lengua de señas colombiana

Después de un análisis de las diversas técnicas disponibles para llevar a cabo la traducción del español a la glosa de lengua de señas colombiana, los cuales se mencionan en los capítulos anteriores, donde se hizo énfasis en tres herramientas diferentes: el uso de redes neuronales recurrentes, específicamente el tipo de red Long short-term memory (LSTM) en el marco de TensorFlow, así como la aplicación de modelos basados en Transformer implementados en Keras.

Gracias al análisis que se realizó sobre el uso de modelos basados en Transformers, como los implementados en la biblioteca Hugging Face, los cuales representan una de las últimas tendencias en el campo del procesamiento del lenguaje natural (NLP), donde se ha demostrado una gran capacidad para capturar relaciones semánticas y sintácticas en el texto, lo que los hace ideales para tareas de traducción y generación de texto (Hugging Face, 2019). La biblioteca Hugging Face proporciona implementaciones preentrenadas de una amplia variedad de modelos basados en Transformers, así como una API fácil de usar que permite a los desarrolladores aprovechar estos modelos para diversas tareas de NLP, incluida la traducción de idiomas (Microsoft Learn, 2024). Tras evaluar las capacidades, limitaciones y necesidades del proyecto con cada uno de estos métodos, se determinó que el modelo más apropiado para la implementación final del prototipo sería el denominado `sentence_similarity_spanish_es` que se encuentra disponible en Hugging Face (Hugging Face, 2019).

Este modelo, diseñado para analizar contextos y hallar la similitud entre dos oraciones, permite abordar la parte de traducción del español a la glosa de lengua de señas del prototipo. Por lo que `sentence_similarity_spanish_es` se convirtió en el núcleo central de la solución propuesta. En su funcionamiento, al recibir una oración en español, ya sea ingresada mediante texto o voz, se establece una conexión con un servidor desarrollado en Python. En este servidor se encuentra alojado el modelo `sentence_similarity_spanish_es`, el cual se encarga de comparar el contexto semántico de la oración de entrada con las frases almacenadas en un diccionario previamente establecido (Hugging Face, 2019).

La tarea principal del servidor consiste en determinar cuál de las frases del diccionario presenta una mayor similitud con la oración de entrada. Este proceso se lleva a cabo mediante

técnicas de comparación de vectores semánticos o similares. Una vez identificada la frase con mayor similitud, se le asigna su correspondiente glosa de lengua de señas.

La glosa obtenida se transmite de vuelta a la aplicación cliente, la cual se ejecuta en un entorno Android. En esta aplicación, se realiza un análisis detallado de la glosa recibida. Se lleva a cabo un proceso de validación para determinar qué animaciones contenidas en el archivo GLB se corresponden con cada uno de los elementos de la glosa. Este archivo GLB contiene una serie de animaciones que representan las diferentes señas del lenguaje de señas que se encuentran en el diccionario previamente establecido, el cual está en formato JSON y abarca señas de emergencia.

Una vez determinadas las animaciones pertinentes, se procede a renderizarlas en un avatar 3D dentro de la aplicación Android. De esta manera, se logra una representación visual dinámica que intenta realizar una buena interpretación de las señas correspondientes a la glosa previamente obtenida. Este enfoque no solo facilita la comprensión de la glosa de lengua de señas, sino que también ofrece una experiencia interactiva y accesible para los usuarios objetivo de la aplicación.

Conclusiones

En el abordaje de la tarea de traducción del español a la glosa de la lengua de señas colombiana (LSC), se exploraron diversas técnicas y herramientas de procesamiento de lenguaje natural (NLP). Inicialmente, se evaluaron plataformas ampliamente reconocidas como Freeling y NLTK, pero presentaron limitaciones significativas para manejar las características únicas de la glosa de la LSC, resultando en traducciones imprecisas y poco naturales. Ante estos desafíos, se implementaron técnicas más avanzadas como la tokenización y el análisis morfosintáctico y semántico, cruciales para descomponer y comprender la estructura gramatical de las oraciones en español y su relación con la glosa.

En este contexto, se desarrollaron modelos de aprendizaje automático como redes neuronales recurrentes (RNN) y el uso del modelo Keras_Transformer. Estas técnicas demostraron capacidad para aprender patrones y traducir oraciones presentes en el corpus de entrenamiento. Sin embargo, su desempeño se vio comprometido al enfrentarse a oraciones nuevas o que diferían significativamente de los datos de entrenamiento, lo que evidenció la necesidad de contar con recursos más amplios y representativos, como corpus de datos más extensos y específicos para la LSC.

Posteriormente, el modelo `sentence_similarity_spanish_es` de Hugging Face, basado en la arquitectura Transformer, demostró ser una opción más efectiva al analizar la similitud semántica entre oraciones y encontrar las glosas más adecuadas a partir de un diccionario preestablecido. No obstante, al no ser un enfoque de traducción directa, podría presentar imprecisiones en ciertos casos, especialmente cuando las oraciones de entrada difieren significativamente de las frases en el diccionario. Estos resultados resaltan la necesidad de continuar explorando y desarrollando modelos de traducción más avanzados y específicos para mejorar la precisión y naturalidad de la traducción del español a la glosa de la LSC.

En cuanto a la selección de herramientas de programación, se eligió Python para el backend, aprovechando su flexibilidad y amplia gama de bibliotecas de NLP. Para el frontend en dispositivos Android, se utilizó inicialmente Java, pero debido a limitaciones en la integración de animaciones 3D, se migró a Kotlin, un lenguaje más adecuado para este propósito, junto con Android Studio como entorno de desarrollo integrado. La elección de una arquitectura cliente-servidor permitió separar eficientemente el procesamiento de la traducción

en el servidor, utilizando Python y las bibliotecas de NLP, de la presentación visual en el dispositivo móvil, facilitando la escalabilidad y el mantenimiento del sistema. Sin embargo, el cambio de Java a Kotlin para el frontend demuestra la importancia de mantener una mentalidad flexible y adaptar las tecnologías según las necesidades del proyecto.

En la búsqueda de la herramienta más adecuada para la creación y animación del avatar, se realizó un análisis exhaustivo de diferentes opciones. Inicialmente, se optó por DAZ Studio debido a su especialización en modelado y animación de personajes humanos, pero se enfrentaron desafíos de compatibilidad al intentar exportar los archivos en el formato requerido. Ante estas dificultades, se eligió Blender, un software de código abierto ampliamente reconocido en animación y modelado 3D, que cumplió con los requisitos necesarios para modelar y animar el avatar destinado a la aplicación móvil. Además, se incorporó Mixamo, un complemento que facilitó la animación del avatar. Esta experiencia resalta la importancia de realizar una investigación exhaustiva y estar abiertos a cambiar de herramienta si las circunstancias lo requieren.

En el desarrollo del prototipo de avatar, se elaboraron aproximadamente 12 animaciones que representan cada palabra de la glosa para las señas de emergencia en la LSC, utilizando Blender y Mixamo. Estas animaciones capturaron con precisión los movimientos manuales requeridos, asegurando una representación fiel de la LSC. Sin embargo, el hecho de animar cada palabra por separado generó cortes visuales al cambiar entre animaciones, lo que puede afectar la fluidez y naturalidad de la interpretación. Esto sugiere la necesidad de explorar enfoques alternativos, como la animación de oraciones completas o la implementación de técnicas de suavizado de transiciones entre animaciones.

Es importante destacar que, si bien se lograron avances significativos en el desarrollo del prototipo, existen áreas que requieren mayor investigación y desarrollo. Los resultados obtenidos con los diferentes enfoques de traducción evaluados, como las redes neuronales recurrentes, el modelo Keras_Transformer y el modelo sentence_similarity_spanish_es de Hugging Face, presentaron limitaciones y desafíos que deben ser abordados para mejorar la precisión, naturalidad y accesibilidad del prototipo de avatar traductor de señas de emergencia en la lengua de señas colombiana.

Finalmente, estas conclusiones analíticas resaltan las lecciones aprendidas durante el proceso, las limitaciones existentes y las oportunidades de mejora. Esto permitirá abordar de

manera más efectiva los desafíos y continuar avanzando en el desarrollo de soluciones tecnológicas accesibles y eficientes que promuevan la inclusión de la comunidad sorda en situaciones de emergencia.

Recomendaciones

Se recomienda enfocar esfuerzos en la recopilación y construcción de un corpus de datos más extenso y representativo de la lengua de señas colombiana (LSC). Contar con un conjunto de datos más diverso y específico para la LSC permitirá mejorar el rendimiento de los modelos de traducción basados en aprendizaje automático, como las redes neuronales recurrentes (RNN) y el modelo Keras_Transformer. Esto contribuirá a incrementar la precisión y naturalidad de las traducciones al enfrentarse a oraciones nuevas o que difieran significativamente de los datos de entrenamiento iniciales.

Se sugiere investigar y desarrollar técnicas de animación más sofisticadas para el avatar, con el objetivo de lograr una representación más fluida y natural de las señas. Esto podría implicar la exploración de enfoques como la animación de oraciones completas en lugar de animaciones individuales por palabra, o la implementación de técnicas de suavizado de transiciones entre animaciones. Estas mejoras en la animación del avatar contribuirían a una interpretación más accesible y comprensible de la LSC.

Se recomienda promover y fortalecer la colaboración interdisciplinaria entre expertos en diferentes áreas, como lingüistas, especialistas en lengua de señas, desarrolladores de software, animadores y diseñadores de interfaces. Esta colaboración cercana podría aportar perspectivas valiosas y conocimientos especializados para abordar los desafíos de manera más integral.

Referencias

- Android Developers. (2017, mayo). *Desarrolla apps para Android con Kotlin*.
<https://developer.android.com/kotlin/stories?hl=es>
- Blancarte, O. (2014, julio). *Que es Service-oriented architecture (SOA)*.
<https://www.oscarblancarteblog.com/2014/07/23/que-es-service-oriented-architecture-soa/>
- Chirivella, A. (2022, febrero). *Blender, qué es y para qué se utiliza*.
<https://www.profesionalreview.com/2022/02/20/blender-que-es-y-para-que-se-utiliza/>
- Harris, C. (2022, mayo). *Arquitectura de microservicios*.
<https://www.atlassian.com/es/microservices/microservices-architecture>
- Hugging Face. (2019, septiembre). *Transformers*.
<https://huggingface.co/docs/transformers/index>
- Ideasti. (2019, mayo). *iOS vs Android, por la supremacía para el desarrollo de nuestra app móvil*. <https://ideasti.co/blog/ios-o-android/>
- IONOS. (2023, enero). *¿Cómo funciona el modelo cliente-servidor?*.
<https://www.ionos.es/digitalguide/servidores/know-how/modelo-cliente-servidor/>
- Likarajo. (2021, abril). *language translation*. https://github.com/likarajo/language_translation
- Maldonado, R. (2024, abril). *Ventajas y desventajas de Python*.
<https://keepcoding.io/blog/ventajas-y-desventajas-de-python/>
- Nova. (2023, marzo). *From RNNs to transformers: The evolution of NLP models*.
<https://aitechtrend.com/from-rnns-to-transformers-the-evolution-of-nlp-models/>
- Santander, C. (2021, abril). *Transformers: Redes Neuronales*.
<https://www.linkedin.com/pulse/transformers-redes-neuronales-cristian-santander/?originalSubdomain=es>
- Sotaquirá, M. (2020). *Traductor con redes Transformer: Traducción de texto de Inglés a Español usando Redes Transformer*.
[https://github.com/codificandobits/Traductor con redes Transformer](https://github.com/codificandobits/Traductor_con_redes_Transformer)
- Wagner, A. (2023, julio). *Arquitectura Monolítica*. <https://es.linkedin.com/pulse/arquitectura-monol%C3%ADtica-ariel-alejandro-wagner>

Williamson, L. (2022, diciembre). *4 ways mixamo makes animation easier*.
<https://www.schoolofmotion.com/blog/4-ways-mixamo-makes-animation-easier>