TRABAJO DE FIN DE MÁSTER

2023-2024

¿SUEÑAN LOS CHATBOTS CON HUMANOS PARLANTES? CREACIÓN DE UN BOT DE TRADUCCIÓN PARA TELEGRAM

MÁSTER EN TRADUMÁTICA: TECNOLOGÍAS DE LA TRADUCCIÓN

FACULTAD DE TRADUCCIÓN E INTERPRETACIÓN

Alexandra Sandino Campos

1529422

TUTOR

Marc Riera Irigoyen

Barcelona, 24 de mayo de 2024

Datos del TFG / *Dissertation data*

Título: ¿Sueñan los chatbots con humanos parlantes? Creación de un bot de traducción para Telegram

Títol: ¿Somien els chatbots amb humans parlants? Creació d'un bot de traducción per a Telegram

Title: Do Chatbots Dream of Talking Humans? Creation of a translation bot for Telegram

Autor/a: Alexandra Sandino Campos

Autor/a: Alexandra Sandino Campos

Author: Alexandra Sandino Campos

Tutor: Marc Riera Irigoyen

Tutor: Marc Riera Irigoyen

Tutor: Marc Riera Irigoyen

Centro: Universidad Autónoma de Barcelona

Centre: Universitat Autónoma de Barcelona

Centre: Autonomous University of Barcelona (UAB)

Estudios: Máster oficial en Tradumática: Tecnologías de la Traducción

Estudis: Màster oficial en Tradumàtica: Tecnologies de la Traducció

Studies: Official master's degree in Tradumatics: Translation Technologies

Palabras clave / Keywords

Traducción automática, Chatbot, Telegram, Programación

Traducció automática, Chatbot, Telegram, Programació

Machine Translation, Chatbot, Telegram, Coding

Resumen del TFM / *Abstract*

El presente trabajo tiene como objetivo el desarrollo de un bot de traducción para la aplicación de mensajería instantánea Telegram mediante el empleo de la API del motor de traducción automática basada en reglas Apertium. Así, se comenzará por una breve exposición de la traducción automática, abordando su definición, su historia, su clasificación y su estado actual. Asimismo, se elucubrará sobre sus potenciales aplicaciones e innovaciones futuras, tomando como referencia los recientes avances en el campo del aprendizaje automático a nivel comercial. A continuación, se llevará a cabo un análisis del proceso de programación del código en las diferentes fases de diseño. Finalmente, se procederá a revisar los objetivos planteados inicialmente, las mejoras implementadas y las posibilidades incorporaciones o funcionalidades futuras.

El present treball té com a objectiu el desenvolupament d'un bot de traducció per a l'aplicació de missatgeria instantània Telegram mitjançant l'ús de la API del motor de traducció automàtica basada en regles Apertium. Així, es començarà per una breu exposició de la traducció automàtica, abordant la seva definició, la seva història, la seva classificació i el seu estat actual. Així mateix, s'elucubrarà sobre les seves potencials aplicacions i innovacions futures, prenent com a referència els recents avanços en el camp de l'aprenentatge automàtic a nivell comercial. A continuació, es durà a terme una anàlisi del procés de programació del codi en les diferents fases de disseny. Finalment, es procedirà a revisar els objectius plantejats inicialment, les millores implementades i les possibilitats incorporacions o funcionalitats futures.

The aim of this research is to develop a translation bot for the instant messaging application Telegram by using the API of the rule-based machine translation engine Apertium. Thus, a brief presentation of machine translation will commence, addressing its definition, history, classification, and current status. Furthermore, its potential applications and future innovations will be discussed, taking as a reference the recent advances in the field of machine learning at a commercial level. Subsequently, an analysis of the programming process of the code in the different design phases will be carried out. Lastly, a review will be conducted of the initial objectives, the improvements implemented and the possible future incorporations or functionalities.

Aviso legal / *Legal notice*

ÍNDICE

ÍNDICE DE FIGURAS

I. Introducción y justificación

Es indiscutible que nos adentramos cada vez con más rapidez en una era intrínsecamente digital, la vida del ser humano medio no se podría concebir sin el uso de las tecnologías en cada paso que da. Así, se podría afirmar que Internet se han convertido en el nuevo «mejor amigo del hombre». Y uno de sus servicios primordiales son las aplicaciones de mensajería instantánea para dispositivos, en especial para teléfonos inteligentes.

Aplicaciones como WhatsApp, Facebook Messenger, Telegram y WeChat han transformado por completo la manera en la que nos comunicamos a nivel mundial. Han facilitado las conexiones globales, convirtiendo lo que eran transacciones parsimoniosas en acciones instantáneas y en tiempo real.

La omnipresencia actual de estas las ha convertido en herramientas imprescindibles que sitúan el mundo entero al alcance de la mano de cualquier persona, ofreciendo medios de comunicación rápidos, sencillos y eficientes. Aun así, pese a los continuos esfuerzos de las empresas por optimizar e innovar, la diversidad lingüística sigue siendo un terreno hostil. No obstante, la traducción automática es el arma que blandir para superar este obstáculo.

Es cierto que los avances e innovaciones en ámbitos como la inteligencia artificial evidencian que existirá una reconfiguración de lo que se considera posible llevar a cabo por una máquina y la continua e irrefutable evolución llevará a que se reconsideren los límites preestablecidos de la traducción automática, lo que abrirá la puerta a un sinfín de aplicaciones necesarias y de gran utilidad que se verán implementadas en la interactividad. Sin embargo, todavía nos encontramos con áreas sin explorar que aportarían al usuario una experiencia mucho más agradable y pragmática.

Luego, la concepción del siguiente trabajo proviene de un interés personal por la traducción automática aplicada a la mejora del día a día de los usuarios y del reflejo de un contexto en el que este tipo de aplicaciones está vigente y existe una necesidad real: por ejemplo y en el caso que nos ocupará, aunque en algunas aplicaciones como WeChat, la aplicación más extendida en países asiáticos como China, sí que se ha introducido la posibilidad de traducir todos los mensajes; el gigante de la mensajería automática europea, WhatsApp, no ha ofrecido todavía esta opción.

II. Objetivos e hipótesis

Como hemos mencionado en el apartado anterior, se parte de la premisa de que existe una necesidad real de solucionar el hecho de que, en su gran mayoría, las aplicaciones de mensajería no ofrecen la opción de traducir de forma automática los mensajes que se envían o se reciben.

Por ende, el objetivo principal del proyecto es la concepción y ejecución de un chatbot totalmente funcional y operativo que permita el uso de traducción automática a los usuarios. En su fase inicial, la aspiración es conseguir que el usuario pueda interactuar con el chatbot y que este genere la traducción de los mensajes que recibe. Sin embargo, se contempla la posibilidad de dar un paso más allá y llegar a desarrollar con éxito la opción de traducir mensajes entre usuarios de un mismo grupo.

Con el fin de asegurar la viabilidad y eficacia del bot, se llevará a cabo un exhaustivo y riguroso proceso de desarrollo con las pruebas diversas en las fases correspondientes. Además, se analizarán los resultados obtenidos a través de los diferentes contextos y fases de programación con el objetivo de recopilar una amplia variedad de datos que permitan aportar un abanico de conclusiones.

Se ha optado por llevar a cabo el proyecto en Telegram debido a que, aunque el código del servidor no está abierto, las aplicaciones y la API sí son de código abierto, lo que permite la integración de bots desarrollados por terceros. Esto significa que cualquier persona puede crear un bot y agregarlo a la plataforma. La decisión de programar en Python se debe a la versatilidad, sencillez y pragmatismo que lo caracteriza y, también, para aprovechar y sacar partido de los conocimientos que se imparten en el máster sobre este lenguaje de programación en concreto. A continuación, se creará una interfaz gráfica atractiva e intuitiva para el usuario a través de JavaScript.

Vemos relevante destacar que el enfoque del proyecto no se centra en un análisis de la calidad de la traducción automática, sino en conseguir lanzar una herramienta funcional y de utilidad que facilite la comunicación en entornos multilingües a través de la aplicación de mensajería instantánea escogida. Luego, no se evaluarán los resultados desde un punto de vista cualitativo ni se llevarán a cabo métricas de calidad de traducción. Por consiguiente, se optará por el uso de la API de Apertium como motor de traducción, debido a su carácter gratuito y de libre acceso.

III. Marco teórico

1. Traducción automática

1.1 Conceptualización de la TA

La traducción automática (TA) ha surgido como una de las ramas más prominentes dentro del ámbito de la lingüística computacional y la inteligencia artificial (Mondal et al., 2023). Además, se considera que constituye la primera aplicación informatizada del procesamiento del lenguaje natural. La disciplina integra diversas áreas de conocimiento provenientes de campos muy diversos, tales como la lingüística, el procesamiento de lenguaje natural, la computación multilingüe, la programación y la ingeniería de software (Bhattacharyya, 2015).

Según Ginestí y Forcada (2009), la traducción automática se define como el proceso mediante el cual un sistema informático, compuesto por ordenadores y programas, traduce textos informatizados escritos en una lengua de origen a textos informatizados escritos en una lengua meta. Su definición pone de manifiesto la importancia de que los textos estén informatizados, es decir, que el formato del texto pueda ser procesado por un ordenador.

Así, el diccionario Harper Collins (s.f.) la define como la producción de un texto en un lenguaje natural desde otro mediante procedimientos informáticos. Por el contrario, la importancia en esta acepción recae sobre que se trata de la confección de un texto en lengua natural, es decir, que una máquina sea capaz de generar un texto que se adecúe al lenguaje humano.

Aunque, como se expondrá a continuación, la historia de la TA se remonta al siglo XX, la reciente globalización y sus implicaciones han puesto de manifiesto la urgente necesidad de traducciones rápidas, así como el volumen masivo de textos a traducir, lo que ha hecho que la automatización de este proceso se vuelva tanto inevitable como imperativa.

1.2 Breve repaso de la historia de la TA

Si bien es posible remontarse al siglo XVII de la mano de renombrados filósofos como Descartes o Leibniz que expresaron sus precoces inquietudes y cavilaciones sobre una traducción automática en su estado más primitivo (Hutchins y Somers, 1992), hasta el siglo XX no se pueden encontrar estudios fehacientes de los primeros esfuerzos por poner en práctica la idea.

En los años treinta, concretamente en 1933, George Artsrouni y Petr Smirnov-Troyanskii fueron los pioneros encargados de presentar dos patentes de sistemas que buscaban una traducción semiautomatizada (Hutchins y Somers, 1992).

El ingeniero franco-armenio Artsrouni presentó una patente de un cerebro mecanizado. Su objetivo era la fabricación de un artefacto que almacenase y recuperase información de forma automática. Pese a que el diseño no estaba destinado al ámbito de la traducción ni se trataba de un lingüista, concibió el primer sistema automático basado en diccionarios multilingües (Poibeau, 2017).

A su vez, el profesor ruso Smirnov-Troyanskii patentó un aparato que seleccionaba y codificaba palabras con el fin de traducirlas. Aunque no se llegó a llevar a término ni a nivel de prototipo, Hutchins y Somers (1992) consideran su propuesta de gran relevancia debido a que se concebía no solo como una codificación de una lengua y su traducción, si no que se tenía en cuenta el contexto sintáctico.

El contexto de la Segunda Guerra Mundial impidió que se materializase el interés en estos dos inventos y se volvieron obsoletos. La Guerra Fría y la aparición de los primeros ordenadores dieron lugar a una época en la que apareció la necesidad de traducción automática. Los recursos limitados disponibles condicionaron los intentos de desarrollo de modelos, siendo solo posibles los enfoques pragmáticos (Poibeau, 2017).

La idea de que un posible uso de los ordenadores fuese el de la traducción la propuso Andrew Booth (Booth, 1958). Según Hutchins y Somers (1992), Booth junto al criptógrafo Warren Weaver, considerado el padre de la TA y el procesamiento del lenguaje natural, debatieron sobre esta posibilidad. Posteriormente, Weaver publicó un memorándum en 1949 que suscitó un gran interés, al que siguió una activa labor de investigación (Bar-Hillel, 1952). Su importancia radica en que la influencia fue tanto científica como política, ya que no solo desarrolló estudios innovadores, sino que también poseía vínculos con organizaciones que ofrecieron financiación (Poibeau, 2017).

En cuestión de unos años, múltiples universidades de EE.UU. empezaron a investigar sobre TA (Hutchins, 1999) y, en 1951, el matemático y lingüista israelí Yehoshua Bar-Hillel en el MIT se convirtió en el primer profesional dedicado únicamente a su investigación (Hutchins y Somers, 1992). En 1954, tuvo lugar la primera demostración pública de su viabilidad, fruto de la colaboración de la Universidad de Georgetown y el IBM (Hutchins, 1999). Su impacto se

tradujo en un aumento de la financiación y de la atención pública. Además, logró captar el interés de la URSS, que inició sus propias investigaciones al año siguiente (Poibeau, 2017).

Una parte considerable de la labor de investigación en esta época fue de suma importancia para el desarrollo de la TA y de los campos de la lingüística computacional y la inteligencia artificial. No obstante, el objetivo fundamental de crear sistemas capaces de generar traducciones de calidad no se consiguió. Pese al optimismo inicial, la desilusión se acentuó a medida que se evidenciaba la complejidad de los problemas que entrañaba (Hutchins y Somers, 1992).

En 1960, Bar-Hillel presentó un informe que reflejaba estos problemas, los decepcionantes resultados de las investigaciones y que los enfoques eran demasiado optimistas. Las dudas que expresó afectaron a los patrocinadores y causaron que muchos profesionales abandonaran el sector, lo que llevo a la solicitud del informe para el Comité Asesor de Procesamiento Automático del Lenguaje (ALPAC, por sus siglas en inglés) en 1966 (Poibeau, 2017).

El informe ALPAC señalaba que la TA era más lenta, menos precisa y dos veces más cara que la traducción humana. Argumentaba que no era necesario continuar con la financiación y sugería que, en su lugar, se desarrollaran programas de automatización de la traducción para traductores humanos (Hutchins y Somers, 1992). Según el informe, el campo carecía de interés práctico, ya que no existían sistemas adecuados para realizar la tarea (Poibeau, 2017).

Como afirma Hutchins y Somers (1992), su repercusión es innegable y el resultado fue la interrupción de la cuantiosa financiación en Estados Unidos. En la década posterior, la investigación se desarrolló en su mayoría en el resto del mundo y empezaron a aparecer en el mercado los primeros sistemas comerciales.

En Canadá, el bilingüismo dio pie a una demanda de traducciones del inglés al francés que superó la capacidad del mercado (Hutchins y Somers, 1992). Por ende, se incentivó la investigación, ya que la necesidad de producción conllevaba unos costes elevados (Poibeau, 2017). En 1976, un grupo de investigación de Montreal desarrolló el sistema Météo para la traducción de predicciones meteorológicas de emisión diaria (Hutchins, 2007). El sistema estuvo en funcionamiento hasta 2002 y tradujo cientos de miles de previsiones (Poibeau, 2017).

En Europa, la Comisión Europea exigía traducciones de documentos científicos, técnicos, administrativos y jurídicos desde y hacia todas las lenguas que formaban parte de ella

(Hutchins y Somers, 1992). La CE decidió instalar el sistema Systran en 1976 (Hutchins, 2007), estableciendo un acuerdo de colaboración que duró toda la década de 1980 (Poibeau, 2017).

A finales de los setenta, resurgió la investigación en EE. UU. con el desarrollo de los sistemas SPANAM y ENGSPAN por la OPS y del sistema METAL, creado por las Fuerzas Aéreas estadounidenses y con el apoyo de Siemens. En Europa, entre 1970 y 1992, se financió el ambicioso proyecto EUROTRA que abarcaba todos los idiomas de la CE. Más tarde, se desarrollaron diversos sistemas basados en este proyecto, como PaTrans en Dinamarca o CAT2 en Alemania (Quah, 2006).

La investigación se mantuvo a lo largo de los ochenta con el afán de encontrar métodos y técnicas de mayor calidad. La mayor parte se produjo en Japón, a raíz del sistema Mu. Un proyecto muy conocido fue el *Fifth Generation Multilingual Interlingua-based Machine Translation System Project* (1987-1995) del *Centre for International Cooperation for Computerization* (CICC) de Japón junto a otros cuatro países asiáticos (Quah, 2006). Asimismo, las principales empresas niponas iniciaron la comercialización de sistemas de TA con programas como Pensee de OKI, HICATS de Hitachi y MeltranJ-E de Mitsubishi (Sánchez Ramos y Rico Pérez, 2020).

En esta década, los avances en lingüística computacional permitieron el desarrollo de enfoques más sofisticados. La mayoría de los productos se trataba de *software* de traducción asistida por ordenador (TAO) entre el japonés y el inglés (Quah, 2006). Además, se desarrollaron sistemas de TA directos, interlingua y de transferencia (Hutchins, 2003). Los sistemas directos más conocidos fueron Systran, Logos y Atlas y los de transferencia fueron Ariane y EUROTRA (Quah, 2006).

La década de los noventa está marcada por un cambio inaudito en las relaciones internacionales. Las telecomunicaciones globales e Internet dieron lugar a nuevas necesidades de traducción para diferentes audiencias (Quah, 2006). La potencia y asequibilidad de los ordenadores comerciales permitieron utilizar los sistemas que antes solo estaban al alcance de los grandes ordenadores (Hutchins, 2007).

La mayoría de las investigaciones hasta finales de los ochenta se centraban en métodos basados en reglas. No obstante, desde 1989, este predominio se vio contrarrestado por la aparición de los basados en corpus (Hutchins, 2007). Estos nuevos métodos supusieron un punto de inflexión. Por ejemplo, animados por el éxito en reconocimiento de voz, en 1994, un grupo del IBM publicó los resultados de los experimentos con el sistema Candide, basado en

métodos puramente estadísticos (Hutchins, 1999). La TA estadística se inspiraba en la teoría de comunicación propuesta por Weaver en su memorándum (Quah, 2006).

El final de los noventa y el inicio del nuevo siglo fue una época de vertiginoso crecimiento en el uso de TA y de las herramientas TAO. La relevante repercusión de Internet en la investigación también abrió las puertas a un nuevo medio por traducir. Por ende, se desarrollaron sistemas con el propósito de traducir páginas web, correos electrónicos e, incluso, mensajes en chats (Quah, 2006). Así, Google Translate, el primer servicio gratuito de TA en línea, vio la luz en 2006 y, en 2007, se lanzó el sistema estadístico MOSES, financiado por la UE y de código abierto. El impacto de estos sistemas provocó que la TA se convirtiera en un producto al alcance de todos los públicos (Sánchez Ramos y Rico Pérez, 2020).

Luego, en el siglo XXI, la TA ya se ha convertido en un producto de consumo masivo, siendo conocido y utilizado tanto por profesionales como por el público general. Los avances significativos en la tecnología de aprendizaje profundo se comenzaron a aplicar en el campo. En 2014, se introdujeron modelos de traducción neuronal de extremo a extremo, dando origen a la TA neuronal. A su vez, se propuso un enfoque multilingüe basado en este nuevo tipo de TA, considerado un hito. En 2015, Baidu implementó el primer sistema neuronal a gran escala, lo que marcó un rápido despliegue de la TA neuronal (Wang et al., 2022a).

Asimismo, las multinacionales más importantes la incorporaron progresivamente. Google presentó Google Neural Machine Translation (GNMT) en 2016 (Wu et al., 2016) y, ese mismo año, Microsoft anunció su propio servicio de TA neuronal (Microsoft Translator, 2016). En 2017, Amazon presentó Sockeye, su *framework* basado en Apache MXNet (Hieber y Domhan, 2017) y el equipo FAIR de Facebook anunció que ya se había comenzado la transición a un modelo neuronal (Sidorov, 2017).

1.3 Clasificación de la TA

1.3.1 Traducción automática basada en reglas

La traducción automática basada en reglas se trata del enfoque clásico en TA y fue el protagonista de gran parte de su historia, hasta la década de los ochenta. Los sistemas se basan en diccionarios bilingües automatizados y reglas lingüísticas creadas por profesionales humanos (Yu y Bai, 2015).

Los grandes inconvenientes son que su construcción requiere mucho tiempo por la cantidad de datos lingüísticos que se deben incluir (Ginestí y Forcada, 2009) y el elevado coste que esto

conlleva, ya que se requiere de lingüistas altamente cualificados que redacten las reglas para cada par de lenguas (Kenny, 2022).

Además, si bien ofrecen mejores resultados que otros métodos en pares de lenguas similares o en lenguas minoritarias que carecen de un gran volumen de corpus (Parra Escartín, 2018), en muchos casos es imposible anticipar al completo el conocimiento necesario (Kenny, 2022), ya que el sistema no será capaz de traducir aquellas estructuras lingüísticas de la que no cuente información codificada en las reglas, gramáticas o diccionarios (Parra Escartín, 2018).

Existen tres modelos básicos: directo, de interlingua y de transferencia. Los modelos parten de diferentes premisas lingüísticas, ya que, aunque todos requieren un análisis de la lengua de partida y la síntesis de la de llegada, varía el tipo o la profundidad del análisis y la base desde la que se realiza la síntesis (Yu y Bai, 2015).

1.3.1.1 Modelo directo

Considera que la traducción consiste principalmente en la transferencia léxica directa. Por ende, analiza una frase de la lengua de partida como una secuencia de palabras, extrae los equivalentes léxicos correspondientes y los reordena en la frase de la lengua meta. Así, si bien incluye un análisis morfológico ínfimo y una redistribución del orden de las palabras, el peso recae en los conocimientos lingüísticos almacenados en los diccionarios o las reglas descritas en algoritmos (Yu y Bai, 2015).

1.3.1.2 Modelo de interlingua

Establece la posibilidad de encontrar representaciones intermedias semántico-sintácticas que permiten establecer vínculos neutros y no ambiguos entre lenguas a través de una interlingua, una lengua intermedia (Yu y Bai, 2015).

Por tanto, se descarta la necesidad de transferir de una lengua a otra, ya que la interlingua es capaz de representar el significado de cada enunciado y, a la vez, preservar la suficiente estructura lingüística como para que la lengua meta sea una traducción fiel del original (Trujillo, 1999).

El proceso consiste en el análisis de la lengua de partida, que acabará con la conversión de un texto de origen en representaciones, y la posterior síntesis de la lengua de llegada con la producción de textos meta basados en las representaciones intermedias (Yu y Bai, 2015).

La principal ventaja del sistema es que proporciona un enfoque que requiere una cantidad mínima de desarrollo de sistemas multilingües al permitir traducir a diversas lenguas meta a la vez (Trujillo, 1999). Sin embargo, la complejidad del diseño y desarrollo de una interlingua adecuada resulta evidente, ya que las representaciones deben abarcar diversos y complejos fenómenos lingüísticos (Yu y Bai, 2015). Así, Stein (2013) considera que el desarrollo de una lengua intermedia capaz de almacenar la información esencial y al completo necesaria para cada idioma se trata de una utopía, ya que hasta ahora no se ha conseguido encontrar tal lengua, a pesar de los numerosos intentos que se han llevado a cabo.

1.3.1.3 Modelo de transferencia

La base del modelo es la correspondencia estructural entre lenguas. Por tanto, parte de la premisa de que, si bien no es posible documentar todos los enunciados de un lenguaje natural, sí es factible establecer un número finito de estructuras representativas. Luego, es posible crear un sistema que las recopile junto a sus equivalentes en otras lenguas (Yu y Bai, 2015).

Este modelo opera en tres fases: análisis, transferencia y síntesis. Primero, se produce una representación abstracta de la estructura interna de un enunciado a partir de su análisis. Después, se busca la representación equivalente en la lengua meta y se llevan a cabo los cambios lingüísticos pertinentes. Finalmente, se genera la estructura superficial de la representación equivalente, es decir, la traducción del enunciado a la lengua meta (Yu y Bai, 2015).

Así, estos sistemas requieren módulos monolingües que primero analicen y, posteriormente, generen los enunciados y módulos de transferencia que vinculen las representaciones equivalentes de estas frases en ambas lenguas (Trujillo, 1999).

1.3.1.4 El caso de Apertium

El sistema Apertium se trata de una plataforma de código abierto de TA basada en reglas, que emplea un motor de traducción de transferencia superficial (Forcada, 2009). Además, si se incorporan los datos lingüísticos correspondientes a un par de lenguas necesarios para el proceso de la TA basada en reglas de transferencia, Apertium proporciona los programas y herramientas necesarios para la construcción de un sistema particular (Ginestí y Forcada, 2009).

La plataforma surgió en 2004, financiada por el Ministerio de Industria, Turismo y Comercio del Gobierno de España para el desarrollo de tecnología relacionada con la traducción de los

idiomas nacionales (Forcada, 2015). Desde entonces, ha recibido financiación pública y privada, incluidas contribuciones de la Generalitat de Catalunya o Google Summer of Code (Khanna et al., 2021).

La primera versión íntegra se lanzó en 2005 y se basaba en dos sistemas previos de TA del grupo Transducens de la Universitat d'Alacant: interNOSTRUM y Traductor Universia (Forcada et al., 2011). Pese a que partía de un diseño simple por estar ideada para la traducción de pares de lenguas similares, a partir de su segunda versión en 2006 se implementó un sistema más avanzado que permitió su adaptación a una amplia variedad de idiomas (Armentano-Oller et al., 2007).

En la actualidad, Apertium se ha convertido en una plataforma fundamental para la construcción de sistemas de TA para lenguas con recursos demasiado limitados para disponer de suficientes datos para los basados en corpus (Khanna et al., 2021). Este éxito se atribuye en parte a su naturaleza de código abierto, la sencillez del diseño, una comunidad activa de desarrolladores y una wiki colaborativa que proporciona toda la información necesaria para el desarrollo de nuevos pares de idiomas (Forcada et al., 2011).

La plataforma consta de un motor independiente de la lengua, un formato simple basado en XML para la codificación de los datos lingüísticos y las herramientas pertinentes para la gestión y entrenamiento de los módulos que forman el motor (Forcada, 2015). Así, está compuesto por una serie de módulos dispuestos en cascada (Forcada, 2009):

- El desformateador separa la información de formato del texto.
- El analizador morfológico segmenta el texto en formas superficiales y asigna a cada forma superficial una o varias formas léxicas.
- El desambiguador léxico categorial selecciona uno de los análisis para las palabras ambiguas según el contexto.
- El módulo de transferencia léxica detecta y trata los patrones de las formas léxicas que requieren un tratamiento especial debido a divergencias gramaticales entre idiomas.
- El generador morfológico produce, a partir de la forma léxica de destino, una forma superficial flexionada de forma adecuada.
- El reformateador reintegra la información de formato original.

1.3.2 Traducción automática basada en corpus

En contraste con aproximaciones anteriores, la traducción automática basada en corpus se caracteriza por la integración de vastas bases de datos compuestas por textos paralelos alineados (Quah, 2006). Asimismo, introduce el empleo del aprendizaje automático, partiendo de la premisa del entrenamiento de un sistema informático para que, mediante dichas bases de datos, adquiera de manera automática los conocimientos necesarios para generar traducciones.

Por consiguiente, este enfoque se distingue de las anteriores por su capacidad para ser entrenado a partir de datos o corpus para que lleve a cabo el proceso de traducción de forma automática y autónoma (Kenny, 2022).

Así, la traducción automática basada en corpus se divide en dos tipos: la TA estadística y la TA neuronal.

1.3.2.1 TA Estadística

La traducción automática estadística aborda el proceso de traducción partiendo de la premisa de que cada decisión está basada en probabilidades (Stein, 2013). Así, examina vastos corpus paralelos informatizados generados por humanos para entrenar algoritmos capaces de traducir a través de parámetros de probabilidad (López, 2008).

La gran ventaja de la TA estadística es la posibilidad de crear sistemas operativos independientes de la lengua y que tampoco dependan de los conocimientos de cada lengua y sus fenómenos particulares. Por tanto, habilita a aquellos idiomas que disponen de un gran número de recursos, pero no de suficientes profesionales capacitados para la elaboración de reglas y gramáticas (Stein, 2013).

Estos sistemas constan de tres componentes fundamentales (Parra Escartín, 2018):

- El modelo de lenguaje es el encargado del cálculo de la probabilidad de corrección de una frase en la lengua meta y, a su vez, de la fluidez de la traducción. Su entrenamiento se lleva a cabo a través de ingentes corpus monolingües.
- El modelo de traducción es el responsable de establecer la correspondencia correcta entre lengua de partida y lengua de destino. Su entrenamiento se realiza a través de corpus alineados.

- El decodificador es el encargado de la producción de la traducción mediante la búsqueda, entre todos los resultados generados, de la palabra o frase con un valor probabilístico más alto.

Asimismo, existen dos tipos de sistemas estadísticos que parten de unidades léxicas diferentes: los basados en palabras y los basados en frases (Babhulgaonkar y Bharad, 2017).

Los basados en palabras parten de la palabra como unidad básica de traducción. Así, la premisa es la existencia de permutación léxica, es decir, que una palabra en la lengua de partida tiene una correspondencia en una de la lengua de destino (Stein, 2013). Posteriormente, las correspondencias se reordenan para producir una frase correcta en la lengua meta (Babhulgaonkar y Bharad, 2017). Cada decisión está asociada a una probabilidad y la secuencia de decisiones con mayor probabilidad general se escoge para producir la traducción optima (Liu y Zhang, 2015).

Los basados en frases consideran la frase, una secuencia de palabras consecutiva, como unidad atómica de traducción (Babhulgaonkar y Bharad, 2017). Por ende, a través de la segmentación en secuencias de frases de la lengua de partida, se permutan las frases en la lengua de destino generando la traducción. La ventaja de este modelo es que permite la gestión de fenómenos lingüísticos como las expresiones idiomáticas (Liu y Zhang, 2015).

1.3.2.2 TA Neuronal

La traducción automática neuronal es un tipo de TA basada en corpus que incorpora el aprendizaje profundo en las redes neuronales (Zhao et al., 2023). Estos sistemas de extremo a extremo aprenden los conocimientos lingüísticos necesarios para la traducción directamente de innumerables corpus paralelos utilizados en la fase de entrenamiento (Wang et al., 2022b).

Estos sistemas ofrecen mejores resultados que el resto de los tipos de TA (Sheeren et al., 2021) y no necesitan reglas diseñadas por profesionales humanos (Wang et al., 2022b). Sin embargo, la necesidad de un gran número de datos para su entrenamiento los hace imposibles de implementar para los pares de lenguas con pocos recursos (Pérez-Ortiz et al., 2022).

Su planteamiento se basa en una única gran red de miles de unidades artificiales, semejantes a las neuronas humanas, que cuentan con la particularidad de la activación. Así, las redes neuronales, definidas por este comportamiento, disponen de neuronas con la capacidad de activarse o desactivarse según los estímulos negativos o positivos que reciben de otras y a la intensidad de las conexiones por las que se traspasan estos estímulos (Forcada, 2017).

Los modelos de TA neuronal más habituales cuentan con una estructura que consta de un codificador y un decodificador (Sheeren et al., 2021).

En primer lugar, se procede al procesamiento de la frase original, en la que el codificador recibirá cada palabra y, en consecuencia, la activación del conjunto de neuronas de la red variará con cada una. Una vez finalizado este proceso, el decodificador proporcionará, a través de una serie de pasos, una puntuación asociada a la probabilidad para cada palabra de la traducción (Pérez-Ortiz et al., 2022). Así, la traducción final se construye mediante la elección continua de la mayor probabilidad en cada una de las posiciones (Forcada, 2017).

Desde la aparición de los primeros sistemas, se han concebido varias arquitecturas, como el basado en redes neuronales recurrentes (RNN) con el mecanismo de atención, el de secuencia-a-secuencia basado en redes convolucionales (ConvSeq2Seq) y el basado en redes con mecanismo de autoatención (Transformer), siendo este último el más implementado actualmente (Zhao et al., 2023).

1.4 Realidad actual y expectativas de futuro

En la actualidad, la traducción automática ha alcanzado resultados formidables gracias a la integración de los avances en nuevas tecnologías, como el aprendizaje automático. Sin embargo, las limitaciones vigentes de estas impiden la independencia total de la intervención de los profesionales humanos, utilizándose como herramientas auxiliares. Aun así, el desarrollo constante de la IA y el aprendizaje profundo propician su actualización y mejora continua (Jiang y Lu, 2021).

Además, cabe mencionar que, si bien ha habido un progreso considerable con la implementación de las redes neuronales (Lyu et al., 2024), la aparición de grandes modelos de lenguaje (LLM, por sus siglas en inglés) ha supuesto nuevas posibilidades en la construcción de sistemas de TA (Hendy et al., 2023). Por ende, se especula que el futuro de la disciplina residirá en la investigación de los LLM implementados en la TA. No obstante, estas nuevas tecnologías no están exentas del planteamiento de debates y nuevos desafíos, como las cuestiones relacionadas con la privacidad, la ética y los derechos de autor, que se deberán abordar con urgencia (Lyu et al., 2024).

En este contexto, la aparición del modelo de LLM, ChatGPT, ha supuesto un hito (Hendy et al., 2023). El modelo de transformador generativo preentrenado (GPT, por sus siglas en inglés) se trata de un sistema de IA desarrollado por la empresa estadounidense OpenAI. En su entrenamiento se utilizaron bases de datos masivas con el objetivo de que, a través de su

estructura y su mecanismo de atención, sea capaz de comprender y generar lenguaje natural (Taye, 2023). Así, pese a que su diseño no está enfocado en la traducción, la exposición a diversas estructuras lingüísticas, como las expresiones idiomáticas, le permiten producir traducciones bastante precisas y adecuadas (Kunst y Bierwiaczonek, 2023). La versión más reciente de este sistema, GPT-4, se lanzó el 15 de marzo de 2023 y presenta unas capacidades más potentes que el modelo anterior, GPT-3.5 (Jiao et al., 2023).

Asimismo, existen otras futuras direcciones que ya se comienzan a esbozar sobre el futuro por el que se podría dirigir la TA, como son la traducción interactiva o la multimodal (Lyu et al., 2024).

- La traducción interactiva busca integrar la participación de los usuarios en el proceso de traducción, ya sea a través de la corrección, revisión o aportación activa. Una posible implementación sería mediante sistemas con LLM integrados que permitan una interacción conversacional con interfaces de usuario interactivas.
- La traducción multimodal consiste en la integración de información visual, sonora o cualquier otro medio no textual en el proceso de traducción. El objetivo es mejorar la calidad y precisión de la traducción en ciertos contextos, como la subtitulación audiovisual, el reconocimiento de voz o la interpretación del lenguaje de signos.

2. Chatbots y Telegram

2.1 Definición de chatbot

Un chatbot, abreviatura del término chat robot, se refiere a una aplicación de software que tiene como fin la interacción con humanos mediante el uso del lenguaje natural. En general, esta interacción se lleva a cabo a través de medios escritos y orales (Hasal et al., 2021). Además, su diseño está orientado a emular los patrones de la interacción humana con el propósito de entablar conversaciones entre humanos y máquinas semejantes a las que se establecen de manera orgánica (Adamopoulou y Moussiades, 2020).

Asimismo, es relevante destacar que, pese a ser usual que se utilicen los términos chabot y bot como sinónimos inequívocos, la realidad precisa es que un bot es un programa informático que automatiza procesos, mientras que un chatbot es un subgénero de bot que se centra en automatizar las interacciones conversacionales (Hasal et al., 2021).

En sus inicios, los chatbots se basaban en reglas, lo que implica que encontraban una coincidencia entre la entrada del usuario y un conjunto de reglas con el que generan una

respuesta predefinida a partir de algoritmos de coincidencia de patrones (Adamopoulou y Moussiades, 2020). Luego, funcionaban con independencia al operador humano e intentaban reconocer la consulta del usuario y ofrecer respuestas adecuadas. En caso de que la consulta se encontrase fuera de los conocimientos que poseían, redirigían la conversación hacia parámetros conocidos u operadores humanos. (Hasal et al., 2021).

Sin embargo, los chatbot actuales se basan en el aprendizaje automático y, a través de la evaluación del contexto, proceden a la proposición de respuestas apropiadas. Gracias a los grandes conjuntos de datos con los que se lleva a cabo su entrenamiento, son capaces de hacer frente a una amplia variedad de situaciones conversacionales (Singh et al., 2024).

2.2 Historia de los chatbots

En 1950, Alan Turing fue el artífice de la idea de que un programa informático fuese capaz de mantener una conversación con un humano sin que este pudiese distinguir la artificialidad del interlocutor. Esta idea se materializó con el test de Turing (Adamopoulou y Moussiades, 2020). Este test se trata de una prueba que evalúa la inteligencia de un ordenador según la capacidad de un humano para distinguir si está interactuando con una máquina u otro humano (Khan y Das, 2017).

El primer chatbot, ELIZA, fue desarrollado por el Instituto Tecnológico de Massachusetts (MIT) en 1966. ELIZA era capaz de responder preguntas simples mediante un procedimiento que consistía en la búsqueda de palabras claves en el texto, la detección y asociación de esta palabra clave con reglas internas y la posterior generación de respuestas. Pese a que su capacidad comunicativa era limitada, es considerada la inspiración para el posterior desarrollo de otros programas más avanzados (Hasal et al., 2021).

En 1972, se desarrolló otro chatbot muy conocido, PARRY, que desempeñaba el papel de un paciente con esquizofrenia. Más sofisticado que su predecesora, a PARRY se le atribuía una personalidad propia, contaba con una estructura de control mejorada y las respuestas se consideraban más emocionales. Más tarde, en 1988, se desarrolló el primer chatbot que utilizaba el aprendizaje automático, Jabberwacky, que utilizaba la asociación de patrones contextuales basados en conversaciones previas a la hora de responder (Adamopoulou y Moussiades, 2020).

En 1995, se creó ALICE, el primer chatbot en línea inspirado en ELIZA. Introdujo la tecnología AIML (Artificial Intelligence Markup Language), un lenguaje de programación derivado de XML, que se encarga de la concordancia de patrones para la relación de una entrada de usuario con

una respuesta de la base de datos del chatbot. En 2001, los avances en la tecnología de los bots permitieron el desarrollo de SmarterChild, que fue el primero en aplicarse a tareas cotidianas a través de la retribución de información de bases de datos (Hasal et al., 2021).

El desarrollo del aprendizaje automático supuso un gran avance con la creación de los asistentes de voz personales e inteligentes, integrados en teléfonos inteligentes o altavoces domóticos que se conectan a internet, entienden comandos de voz y generan respuesta de voz: Siri de Apple en 2011, Alexa de Amazon en 2014, Cortana de Microsoft en 2013 y el asistente de Google en 2016 (Adamopoulou y Moussiades, 2020).

Asimismo, Telegram lanzó su propia plataforma de bots en junio de 2015, que permitió a los desarrolladores a crear chatbots integrados en la aplicación (Khan y Das, 2017). Poco después, a principios de 2016, se produjo otra evolución significativa en el aprendizaje automático considerada un hito en la historia de la comunicación entre humanos y máquinas. Muchas plataformas relacionadas con las redes sociales también permitieron la creación de chatbots para sus marcas o servicios con el objetivo de la realización de acciones diarias especificas a través de sus dispositivos o aplicaciones (Adamopoulou y Moussiades, 2020).

2.3 Usos y aplicaciones

La introducción del aprendizaje automático ha impactado en la vida cotidiano a través del diseño de aplicaciones y dispositivos, conocidos como agentes inteligentes, que automatizan procesos del día a día (Adamopoulou y Moussiades, 2020).

Debido a la evolución de Internet, muchas empresas recurren al uso de plataformas en línea para la gestión de la atención al cliente y por eso optan por los chatbots para mejorar la eficacia y calidad de sus servicios, así como para optimizar sus operaciones internas y aumentar la productividad (Suta et al., 2020).

En consecuencia, el público general utiliza los chatbots para diversas actividades, como compras en línea, consultas en páginas web, pedir comida a domicilio, asistencia sanitaria, entre otras muchas acciones cotidianas (Hasal et al., 2021).

Así, si bien los chatbots han formado parte en los últimos años de la nueva era de la aplicación del aprendizaje automático al encargarse de tareas del día a día, se espera que pronto sean capaces de realizar acciones aún más complejas con la aplicación en hogares o vehículos inteligentes (Maher et al., 2020).

2.4 Telegram dentro de la industria de la mensajería instantánea

Como ya se ha mencionado con anterioridad, una de las numerosas plataformas que facilita el desarrollo y la implementación de bots en su entorno es Telegram, la aplicación de mensajería instantánea en la nube. Fundada en 2013 por Pavel Durov, junto a su hermano Nikolai, alcanzaba los 900 millones de usuarios activos en marzo de 2024 y, hoy en día, se ha convertido en una de las más populares a nivel mundial dentro de su ámbito (Murphy, 2024).

2.4.1 Aplicación de los chatbots en Telegram

Así, en junio de 2015, Telegram introdujo Bot API, habilitando a terceros la función de crear bots dentro de la aplicación. Bot API es una interfaz basada en HTTP diseñada para los desarrolladores interesados en crear bots en la aplicación y que los integra con la plataforma (Telegram, 2015).

La aplicación define los bots como cuentas especiales que no requieren de un número de teléfono para su creación, si no que se tratan de la interfaz de un código alojado en el servidor del desarrollador (Telegram, s.f.-a).

Entre algunas de las funciones habilitadas se encuentra la capacidad de realizar pagos desde la plataforma sin ningún tipo de comisión, mediante la integración de diferentes proveedores de pago integrados de terceros introducida en 2017 (Telegram, 2022). Además, permite la creación de herramientas personalizadas que lleven a cabo tareas específicas, la integración con servicios externos como Youtube o Gmail, la recreación o sustitución de páginas webs completas, la creación de juegos, concursos y mucho más (Telegram, s.f.-b).

Asimismo, una de las principales ventajas que ofrece la plataforma es que cuenta con diversas guías detalladas sobre cómo crear un bot, sus funcionalidades, ejemplos de código y artículos que resuelven todas las dudas que puedan surgir a la hora de programar (Telegram, s.f.-a).

IV. Metodología

1. Consideraciones previas

1.1 Docker

Docker se trata de una conocida plataforma de código abierto basada en contenedores, concebida para la elaboración, distribución y ejecución de aplicaciones. Su objetivo es facilitar la generación de contenedores ligeros y portables que permitan la ejecución de aplicaciones en un sistema de cualquier índole con la única condición de instalar el programa. Así, un contenedor en Docker representa un paquete de software completo con todos los recursos necesarios para su ejecución en cualquier máquina, sin requerir una instalación directa en la misma (Docker, s.f.).

Las ventajas principales por las que este servicio se ha convertido en una solución habitual para desarrolladores de todo el mundo incluyen la rapidez en el inicio, los requisitos mínimos de memoria y almacenamiento, la versatilidad para desplegarse en diversos entornos, la agilidad al evitar la descarga manual de dependencias, así como la capacidad de proporcionar un entorno aislado, entre otras.

La elección de Docker para este proyecto se justifica por la necesidad de sortear las dificultades relacionadas con la instalación de la API, los paquetes individuales de pares de idiomas y otras librerías y dependencias requeridas para la creación del bot. Ante esta situación, Docker se presenta como una solución práctica y eficiente.

1.2 BotFather

Previamente a la programación del código, el proceso inicial para crear un bot en Telegram implica acceder a @BotFather (https://t.me/botfather) dentro de la aplicación para su registro y la obtención del token correspondiente. BotFather, que se define a sí mismo como "un bot para gobernarlos a todos", se trata de la herramienta fundamental en la gestión y creación de bots dentro de la plataforma.

Dentro de Telegram, un token representa una cadena de caracteres que tiene como función la autentificación del bot en la API y es único e intransferible. La obtención del token se realiza mediante el uso del comando `/newbot`, siguiendo las instrucciones proporcionadas por el propio BotFather. Estas se presentan de manera clara y detallada, guiando al usuario a través

del proceso de asignar un nombre y un nombre de usuario al bot, y finalmente proporcionando el token necesario, como podemos observar en las Figuras 1 y 2.

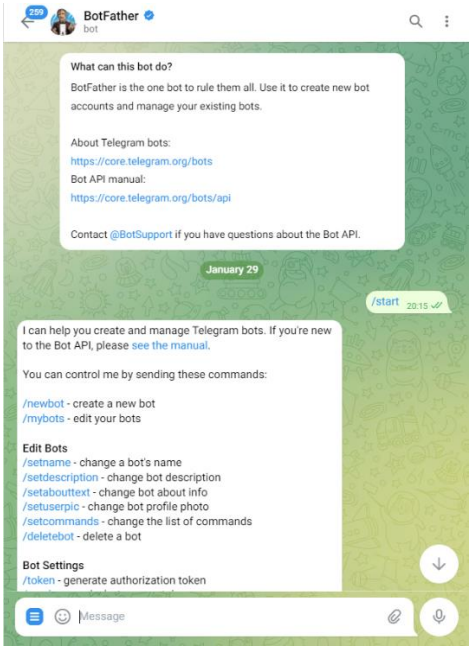

Figura 1: Primera interacción con @BotFather

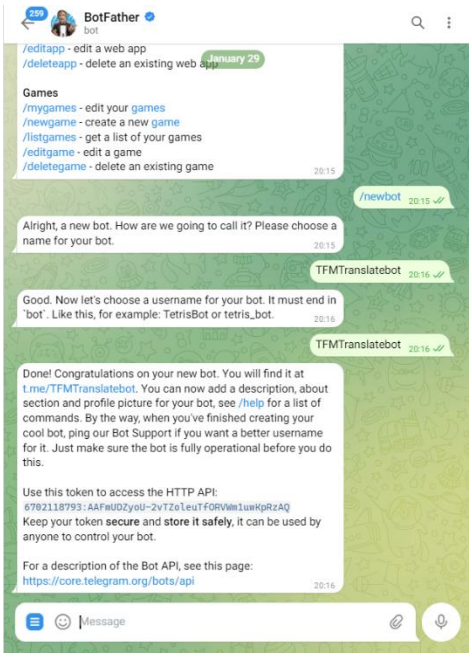

Figura 2: Obtención del token

Una vez obtenido el token, el siguiente paso consistirá en programar y ejecutar el bot desde el sistema operativo del usuario. Una vez en funcionamiento, podrá interactuar con los usuarios a través de la aplicación web o móvil de Telegram.

2. Desarrollo y análisis del código

2.1 Primera propuesta

A continuación, se procede a una explicación detallada de la primera propuesta de código para la configuración del bot que, en su fase inicial, tiene como propósito la generación de traducciones automáticas de cualquier mensaje recibido entre dos idiomas especificados mediante la API de Apertium. El código completo se presenta en la Figura 3 y su implementación en Telegram se muestra en la Figura 4.

```python
BOT_USERNAME = "TFMTranslatebot"
BOT_TOKEN = "6702118793:AAFmUDZyoU-2vTZoleuTfORVWm1uwKpRzAQ"


SERVER_IP = "localhost"
SERVER_PORT = 32769

APERTIUM_QUERY = f"http://{SERVER_IP}:{SERVER_PORT}"

import telebot
import requests
import json

mybot = telebot.TeleBot(BOT_TOKEN)

@mybot.message_handler(func=lambda msg: True)
def reply(message):
    translated_message = translate(source="spa", target="eng",
        message=message.text)
    mybot.reply_to(message, translated_message)


def translate(source, target, message):
    myquery = f"{APERTIUM_QUERY}/translate?langpair=
    {source}|{target}&q={message}"
    response = requests.get(myquery)
    response = json.loads(response.text)
    translation = response["responseData"]["translatedText"]
    return translation

mybot.infinity_polling()
```

Figura 3: Código de la 1ª propuesta

Figura 4: 1ª prueba en Telegram

Lo primero que se lleva a cabo es la definición de las variables que contienen la información necesaria para establecer una conexión entre el código del bot, el servidor de Telegram y Docker. La Ip y el puerto se tratan de valores definidos por Docker. Así, la IP siempre se tratará de *localhost*, dado que se lanza desde el mismo ordenador. En caso de que se tratase de un servidor externo, este valor sería diferente. En cambio, el puerto puede variar según la disponibilidad del sistema operativo, siendo establecido por Docker al iniciarse.

```python
BOT_USERNAME = "TFMTranslatebot"
BOT_TOKEN = "6702118793:AAFmUDZyoU-2vTZoleuTfORVWm1uwKpRzAQ"


SERVER_IP = "localhost"
SERVER_PORT = 32769
```

Posteriormente, se construye la consulta al servidor de Apertium utilizando la dirección IP y el puerto específico.

```python
APERTIUM_QUERY = f"http://{SERVER_IP}:{SERVER_PORT}"
```

Dado que el código emplea diferentes bibliotecas para llevar a cabo diversas funciones, se importan las que se implementarán.

```
10  import telebot
11  import requests
12  import json
```

Luego, se crea una instancia de la clase *TeleBot* de la librería homónima que recibe como argumento el token único de acceso para conectar con Telegram.

```
14  mybot = telebot.TeleBot(BOT_TOKEN)
```

Así, se introduce el siguiente decorador que activa la función *reply* cualquier mensaje. Por ende, cada vez que se envía un mensaje al bot, se ejecuta la función *reply* y, a continuación, la función *translate*. A su vez, se establece el idioma de origen como el español y el de destino como el inglés. Después, responderá al mensaje del usuario con la traducción generada.

```
16  @mybot.message_handler(func=lambda msg: True)
17  def reply(message):
18      translated_message = translate(source="spa", target="eng", message=message.text)
19      mybot.reply_to(message, translated_message)
```

La función *translate* toma los parámetros obtenidos anteriormente del idioma de origen, el de destino y el mensaje que traducir. A su vez genera una consulta con ellos y realiza una solicitud al servidor de Apertium. Luego, la función *get* de la librería *request* efectúa una petición HTTP al servidor y devuelve un objeto en formato JSON, que se procesa mediante *json.loads* para convertirlo en un objeto Python que pueda procesarse y manipularse.

Finalmente, se utiliza la función *infinity_polling* de la instancia *TeleBot* para que comience a escuchar los eventos provenientes del servidor de Telegram de manera indefinida. Así, el bot se encuentra en todo el momento a la espera de que el usuario envíe un mensaje.

```
22  def translate(source, target, message):
23      myquery = f"{APERTIUM_QUERY}/translate?langpair={source}|{target}&q={message}"
24      response = requests.get(myquery)
25      response = json.loads(response.text)
26      translation = response["responseData"]["translatedText"]
27      return translation
28
29  mybot.infinity_polling()
```

2.2 Establecimiento de la detección automática del idioma

Tras lograr el objetivo inicial de que el bot generara traducciones, se programa para que fuese capaz de detectar de forma automática el idioma del mensaje que reciba, mediante el uso de la API de Apertium. Se presenta el código en la Figura 5 y la prueba correspondiente en la aplicación en la Figura 6.

```python
BOT_USERNAME = "TFMTranslatebot"
BOT_TOKEN = "6702118793:AAFmUDZyoU-2vTZoleuTfORVWm1uwKpRzAQ"
SERVER_IP = "localhost"
SERVER_PORT = 32769
APERTIUM_QUERY = f"http://{SERVER_IP}:{SERVER_PORT}"

import telebot
import requests
import json

mybot = telebot.TeleBot(BOT_TOKEN)

@mybot.message_handler(func=lambda msg: True)
def reply(message):
    langcode = detect_lang(message=message.text)
    translated_message = translate(source=langcode, target="eng",
        message=message.text)
    mybot.reply_to(message, translated_message)

def detect_lang(message):
    myquery = f"{APERTIUM_QUERY}/identifyLang?q={message}"
    identity = requests.get(myquery)
    identity = json.loads(identity.text)
    return max(identity, key=identity.get)

def translate(source, target, message):
    myquery = f"{APERTIUM_QUERY}/translate?langpair=
    {source}|{target}&q={message}"
    response = requests.get(myquery)
    response = json.loads(response.text)
    translation = response["responseData"]["translatedText"]
    return translation

mybot.infinity_polling()
```

Figura 5: Código de la 2ª propuesta

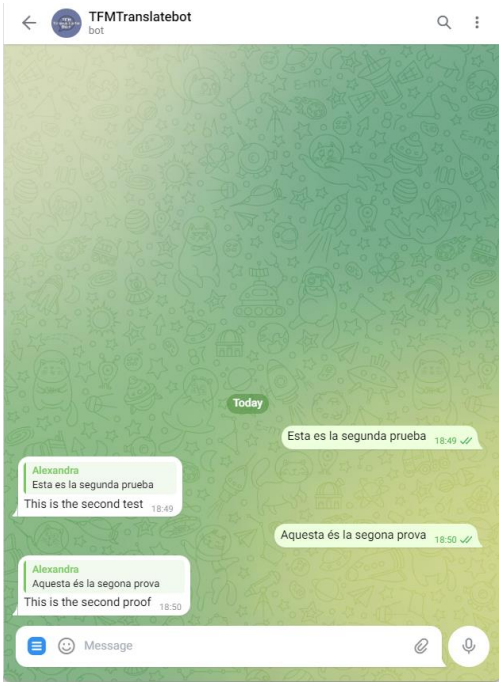

Figura 6: 2ª prueba en Telegram

Así, se utiliza la función *detect_lang*, que lleva a cabo una petición de la misma índole que la función *translate* para determinar el idioma en el que está escrito el mensaje. En primer lugar, se realiza la consulta *identifyLang* a la API de Apertium con el fin de que permita recibir valores probabilísticos asociados a los diferentes idiomas posibles, que se almacenan en la variable *identity*.

```python
def detect_lang(message):
    myquery = f"{APERTIUM_QUERY}/identifyLang?q={message}"
    identity = requests.get(myquery)
```

La respuesta, que se trata de un diccionario en formato JSON, se convierte en un objeto Python mediante la función *json.loads*. Una vez convertido, se determina el idioma concreto a través de la función *max*, que devuelve el código del idioma con el valor más alto como idioma que traducir.

```python
    identity = json.loads(identity.text)
    return max(identity, key=identity.get)
```

Posteriormente, el idioma detectado se almacena en la variable *langcode*, que se establece como idioma de origen en la posterior función *translate.*

```python
    langcode = detect_lang(message=message.text)
    translated_message = translate(source=langcode, target="eng", message=message.text)
    mybot.reply_to(message, translated_message)
```

2.3 Introducción de mensajes automáticos

El siguiente objetivo que cumplir es la mejora de la experiencia del usuario mediante la introducción de mensajes automáticos que aparezcan en el chat de la aplicación. Por ende, se opta por definir mensajes que ayudan y guían al usuario para conseguir un interacción más intuitiva y amena. A continuación, se puede observar la Figura 7, que muestra el código completo, y las Figuras 8 y 9, que muestra los mensajes en el contexto de Telegram.

```python
BOT_USERNAME = "TFMTranslatebot"
BOT_TOKEN = "6702118793:AAFmUDZyoU-2vTZoleuTfORVWm1uwKpRzAQ"
SERVER_IP = "localhost"
SERVER_PORT = 32769
APERTIUM_QUERY = f"http://{SERVER_IP}:{SERVER_PORT}"
import telebot
import requests
import json

mybot = telebot.TeleBot(BOT_TOKEN)

@mybot.message_handler(commands=['start', 'help'])
def start_cmd(message):
    start_msg = "¡Hola!👋 Soy tu asistente de traducción automática 🤖." \
        "¿En qué puedo ayudarte? 🦾" \
        "Solo necesito que me envíes el mensaje que quieres traducir y sabré que idioma es 🧐."
    mybot.send_message(message.chat.id, start_msg)

@mybot.message_handler(func=lambda msg: True)
def reply(message):
    langcode = detect_lang(message.text)
    if langcode is None:
        mybot.reply_to(message, "Lo siento, no hablo ese idioma 🤓.")
    else:
        translated_message = translate(langcode,"eng_US", message.text)
        mybot.reply_to(message, translated_message)

def detect_lang(message):
    myquery = f"{APERTIUM_QUERY}/identifyLang?q={message}"
    identity = requests.get(myquery)
    identity = json.loads(identity.text)
    selected_lang = max(identity, key=identity.get)
    selected_lang_prob = identity[selected_lang]
    if selected_lang_prob < 0.1:
        return None
    else:
        return selected_lang

def translate(source, target, message):
    myquery = f"{APERTIUM_QUERY}/translate?langpair={source}|{target}&q={message}"
    response = requests.get(myquery)
    response = json.loads(response.text)
    translation = response["responseData"]["translatedText"]
    return translation
mybot.infinity_polling()
```

Figura 7: Código de la 3ª propuesta

Figura 8: 3ª prueba en Telegram

Figura 9: 4ª prueba en Telegram

El primer paso es añadir un decorador mediante la librería *TeleBot* que especifica que la función *start_cmd* se ejecuta al recibir los comando */start* o */help.* Por tanto, la función solo se llevará a cabo si el usuario activa uno de los dos comandos a través del chat con el bot. A continuación, la variable *start_msg* contiene el mensaje predefinido como respuesta al comando, que se envía al usuario de forma automática mediante la función *mybot.send_message*.

```
17  @mybot.message_handler(commands=['start', 'help'])
18  def start_cmd(message):
19      start_msg = "¡Hola!👋 Soy tu asistente de traducción automática 🤖." \
20          "¿En qué puedo ayudarte? 💪" \
21          "Solo necesito que me envíes el mensaje que quieres traducir y sabré que idioma es 🧐."
22      mybot.send_message(message.chat.id, start_msg)
```

Además, se configura otro mensaje automático dentro de la función *reply*. Se introduce la expresión condicional *if* que verifica si la variable *langcode* está vacía, es decir, si es igual a *None*. En tal caso, el bot responde con el mensaje definido dentro de la función *mybot.reply_to.* En caso contrario, se procede al mismo proceso que se había programado en la versión anterior del código.

```
26  @mybot.message_handler(func=lambda msg: True)
27  def reply(message):
28      langcode = detect_lang(message.text)
29      if langcode is None:
30          mybot.reply_to(message, "Lo siento, no hablo ese idioma 🤓.")
31      else:
32          translated_message = translate(langcode,"eng_US", message.text)
33          mybot.reply_to(message, translated_message)
```

Asimismo, se ha modificado la función preexistente *detect_lang* para añadir la variable *selected_lang_prob*, que almacena la probabilidad asociada al idioma con el valor probabilístico más alto detectado por la API de Apertium.

Así, si la probabilidad es menor que 0,1, se devuelve *None*. Este umbral se introduce debido a la búsqueda de un modo de mejorar y perfeccionar la detección del idioma para que sea más fiable.

```
33  def detect_lang(message):
34      myquery = f"{APERTIUM_QUERY}/identifyLang?q={message}"
35      identity = requests.get(myquery)
36      identity = json.loads(identity.text)
37      selected_lang = max(identity, key=identity.get)
38      selected_lang_prob = identity[selected_lang]
39      if selected_lang_prob < 0.1:
40          return None
41      else:
42          return selected_lang
```

2.4 Incorporación de menú con opciones

Con la mejora de la experiencia del usuario y una óptima interactividad en mente, se dispuso el objetivo de integrar un menú que incluya los idiomas de destino ejecutables para el idioma de origen detectado en el mensaje del usuario.

Luego, el código íntegro del diseño del menú se plasma en la Figura 10 y su aspecto en la interfaz de la aplicación se observa en las Figuras 11 y 12.

```python
BOT_USERNAME = "TFMTranslatebot"
BOT_TOKEN = "6702118793:AAFmUDZyoU-2vTZoleuTfORVWm1uwKpRzAQ"

SERVER_IP = "localhost"
SERVER_PORT = 32770

APERTIUM_QUERY = f"http://{SERVER_IP}:{SERVER_PORT}"

USER_MESSAGE = ""

import telebot
import requests
import json
from iso639 import languages

mybot = telebot.TeleBot(BOT_TOKEN)

response = requests.get(f"{APERTIUM_QUERY}/listPairs")
response = json.loads(response.text)
LANG_PAIRS_LIST = response["responseData"]
LANG_PAIRS_DICT = {}
for item in LANG_PAIRS_LIST:
    source = item['sourceLanguage']
    target = item['targetLanguage']
    if source in LANG_PAIRS_DICT:
        LANG_PAIRS_DICT[source].append(target)
    else:
        LANG_PAIRS_DICT[source] = [target]

print(LANG_PAIRS_LIST)
print(LANG_PAIRS_DICT)

@mybot.message_handler(commands=['start', 'help'])
def start_cmd(message):
    start_msg = "¡Hola!👋 Soy tu asistente de traducción automática 🤖. ¿En qué puedo ayudarte? 🦾" \
        "Solo necesito que me envíes el mensaje que quieres traducir y sabré que idioma es 🧐."
    mybot.send_message(message.chat.id, start_msg)


@mybot.message_handler(func=lambda msg: True)
def menu(message):
    langcode = detect_lang(message.text)
    if langcode is None:
        mybot.reply_to(message, "Lo siento, no hablo ese idioma 🤬.")
    else:
        target_list = LANG_PAIRS_DICT[langcode]
        menu = telebot.types.InlineKeyboardMarkup(row_width=2)
        for item in target_list:
            try:
                langname = languages.get(part3=item)
            except Exception:
                continue
            element = telebot.types.InlineKeyboardButton(langname.name, callback_data=f"{langcode}|{item}")
            menu.add(element)
        global USER_MESSAGE
        USER_MESSAGE = message.text
        mybot.send_message(message.chat.id, "¿A qué idioma quieres traducir tu mensaje?", reply_markup=menu)

@mybot.callback_query_handler(func=lambda call: True)
def reply(answer):
    chat_id = answer.message.chat.id
    lang_pair = answer.data.split('|')
    translated_message = translate(lang_pair[0],lang_pair[1], USER_MESSAGE)
    mybot.send_message(chat_id, translated_message)

def detect_lang(message):
    myquery = f"{APERTIUM_QUERY}/identifyLang?q={message}"
    identity = requests.get(myquery)
    identity = json.loads(identity.text)
    selected_lang = max(identity, key=identity.get)
    selected_lang_prob = identity[selected_lang]
    if selected_lang_prob < 0.1:
        return None
    else:
        return selected_lang

def translate(source, target, message):
    myquery = f"{APERTIUM_QUERY}/translate?langpair={source}|{target}&q={message}"
    response = requests.get(myquery)
    response = json.loads(response.text)
    translation = response["responseData"]["translatedText"]
    return translation

mybot.infinity_polling()
```

Figura 10: Código de la 4ª propuesta

Figura 11: 5ª prueba en Telegram

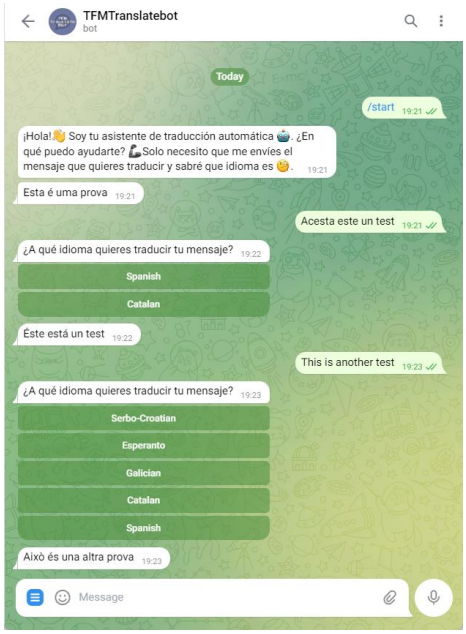

Figura 12: 6ª prueba en Telegram

En primer lugar, se procede a la definición de una variable de tipo global, *USER_MESSAGE*, que tiene como fin almacenar el mensaje que traducir para poder utilizarlo en diferentes funciones. Además, se importa un módulo de una nueva librería, *languages* de *iso639*, para facilitar la gestión y uso de códigos de idioma ISO-639. Se escoge esta librería en concreto porque la API de Apertium utiliza esta norma en su servidor.

```
8   APERTIUM_QUERY = f"http://{SERVER_IP}:{SERVER_PORT}"
9   USER_MESSAGE = ""
10
11  import telebot
12  import requests
13  import json
14  from iso639 import languages
```

A continuación, mediante la librería *requests* se realiza la consulta */listPairs* al servidor de la API de Apertium con el objetivo de obtener la lista de pares de idiomas disponibles. Así, la respuesta con estructura JSON, que contiene una lista de objetos que representan los pares de idiomas, se convierte en un objeto Python manipulable mediante *json.loads* y se almacena en la variable *LANG_PAIRS_LIST*.

Para poder estructurar de manera organizada los pares de idiomas, se opta por crear un diccionario denominado *LANG_PAIRS_DICT*. Por tanto, este diccionario almacena los pares de idiomas con cada idioma de origen como una clave y los de destino correspondientes como valores asociados a las claves pertinentes.

```
18  response = requests.get(f"{APERTIUM_QUERY}/listPairs")
19  response = json.loads(response.text)
20  LANG_PAIRS_LIST = response["responseData"]
21  LANG_PAIRS_DICT = {}
```

Para la creación y estructuración del diccionario, se inicia un bucle *for* que extrae cada elemento de la lista de la variable *LANG_PAIRS _LIST*. Luego, extrae los idiomas de origen y de destino con el propósito de almacenarlos en las variables *source* y *target*, respectivamente.

```
22  for item in LANG_PAIRS_LIST:
23      source = item['sourceLanguage']
24      target = item['targetLanguage']
```

Después, se lleva a cabo una estructura de control condicional *if* que verifica si la variable *source* está en el diccionario y, si es así, se agrega el elemento *target* asociado al idioma de origen mediante la expresión *LANG_PAIRS_DICT[source].append(target)*. Por tanto, se accede a la lista de idiomas de destino vinculados a cada idioma de origen en el diccionario *LANG_PAIRS_DICT* y, a continuación, mediante el método de clase lista *append(),* se añade el idioma de destino al final de la lista del idioma de origen dentro en el diccionario. En cambio, si el idioma de origen no está presente, se ejecuta el bloque de código bajo *else,* que crea una nueva entrada con el idioma de origen como clave y una lista con el idioma de destino como único elemento.

Finalmente, se imprime la lista de idiomas y el diccionario con los pares de idiomas. La finalidad de añadir estas dos funciones es verificar que los datos se procesan de forma adecuada.

```
25      if source in LANG_PAIRS_DICT:
26          LANG_PAIRS_DICT[source].append(target)
27      else:
28          LANG_PAIRS_DICT[source] = [target]
29
30  print(LANG_PAIRS_LIST)
31  print(LANG_PAIRS_DICT)
```

Se establece el decorador *@mybot.message_handler* con el fin de la función *menu* gestione todos los mensajes que recibe el bot. Luego, se alamcena el idioma de destino en la variable *langcode*, obtenido mediante la ejecución de la función *detect_lang*. A continuación, se define la condición if, que verifica si la variable *langcode* está vacía y, en caso de estarlo, responde al usuario con el mensaje configurado mediante el método *mybot.reply_to*.

```
38  @mybot.message_handler(func=lambda msg: True)
39  def menu(message):
40      langcode = detect_lang(message.text)
41      if langcode is None:
42          mybot.reply_to(message, "Lo siento, no hablo ese idioma 😖.")
```

En cambio, si sí se ha podido detectar un idioma, se ejecuta el bloque de código bajo *else*: se accede al diccionario creado con anterioridad, se utiliza *langcode* como la clave, es decir, como el idioma de origen, y se obtiene la lista de idiomas de destino disponibles como valores asociados. Estos idiomas se almacenan en la variable *target_list*.

```
43      else:
44          target_list = LANG_PAIRS_DICT[langcode]
```

A continuación, se procede a la creación del menú de botones en línea con las opciones de los idiomas de destino. El primer paso es definir dentro de la variable *menu* la instancia de la clase *InlineKeyboardMarkup*, que crea un objeto de ese tipo y permite mostrar botones en los mensajes de Telegram. Además, se especifica que debe tener un máximo de dos botones por fila.

```
46          menu = telebot.types.InlineKeyboardMarkup(row_width=2)
```

Después, se inicia un bucle *for* que, mediante el bloque *try,* intenta obtener el nombre completo en inglés del idioma correspondiente al código de tres letras del ISO-639-3, que es el *item*, a través de la función *get* de *languages.* El resultado se almacena en la variable *langname*. Si tienen lugar una excepción, se ejecutará el bloque *except Exception* que tiene establecido que pase a la siguiente iteración. Esto se indica para que el código no deje de funcionar en caso de que no encuentre una traducción de código a nombre completo y, simplemente, continue mostrando el código de tres letras.

```
47          for item in target_list:
48              try:
49                  langname = languages.get(part3=item)
50              except Exception:
51                  continue
```

Posteriormente, se crea el botón, almacenado en la variable *element*, y se establece que el texto visible sea el contenido de la variable *langname*. Además, mediante *callback_data,* se especifica que los datos del botón, al ser presionado por el usuario, se envían al bot. Y *f"{langcode}|{item}* crea una cadena con el código del idioma de origen, en la variable *langcode*, y el de destino, en *item*, separados por una barra vertical. Finalmente, se añade el botón *element* al teclado en la línea *menu* mediante *menu.add(element).*

```
51                element = telebot.types.InlineKeyboardButton(langname.name,
52                    callback_data=f"{langcode}|{item}")
53                menu.add(element)
```

Así, se define *USER_MESSAGE* como una variable global para poder acceder y modificarla fuera de una función concreta. Además, el mensaje original se almacena en esta variable para que otras funciones puedan acceder a él. Finalmente, mediante *mybot.send_message*, se envía el mensaje predefinido al usuario junto con el menú de botones previamente creado utilizando el parámetro *reply_markup*.

```
54            USER_MESSAGE = message.text
55            mybot.send_message(message.chat.id,
56            "¿A qué idioma quieres traducir tu mensaje?", reply_markup=menu)
```

Se define el nuevo decorador *@mybot.callback_query_handler* para indicar que la función *reply* maneja todas las consultas de callback generadas por los botones del menú. Así, se define la función *reply*, que toma *answer* como el argumento representante de la respuesta a la consulta de callback.

```
59    @mybot.callback_query_handler(func=lambda call: True)
60    def reply(answer):
```

A continuación, se extrae el ID del chat y se almacena en la variable *chat_id* para asegurar que el bot de Telegram envía la respuesta traducida al chat correcto, que se identifica por ese ID único. Además, la consulta de callback contiene el par de idiomas entre los que se debe realizar la traducción y, mediante la función *split('|')*, se dividen los datos en una lista con la barra vertical como delimitador. Esta información se almacena en la variable *lang_pair*.

```
61        chat_id = answer.message.chat.id
62        lang_pair = answer.data.split('|')
```

La siguiente variable, *translated_message*, almacena el mensaje traducido mediante una llamada a la función translate, pasando como parámetro el mensaje almacenado en la variable global *USER*_MESSAGE, el primer elemento de la lista *lang_pair* como el idioma de origen y el segundo como el idioma de destino. Finalmente, el bot envía el mensaje ya traducido al chat correcto con a través de la función *mybot.send_message*.

```
63        translated_message = translate(lang_pair[0],lang_pair[1], USER_MESSAGE)
64        mybot.send_message(chat_id, translated_message)
```

2.5 Integración en grupos

Por último, se procede la última mejora del bot: la posibilidad de integrarlo en un grupo y que tenga la capacidad de traducir los mensajes que los usuarios indiquen sin interrumpir una conversación. Varias interacciones con el bot se muestran en las Figuras 13, 14, 15 y 16, y el código se presenta en la Figura 17.

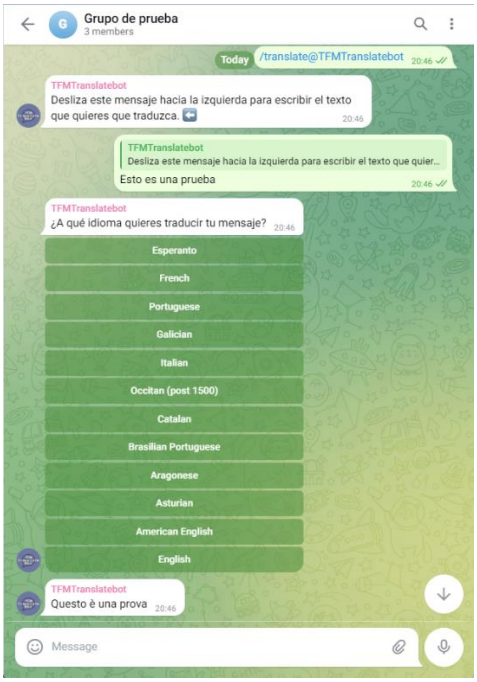

Figura 13: 7ª prueba en Telegram

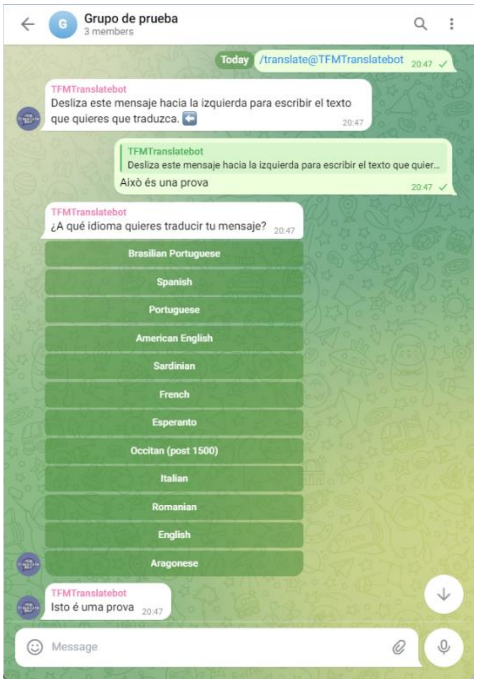

Figura 14: 8ª prueba en Telegram

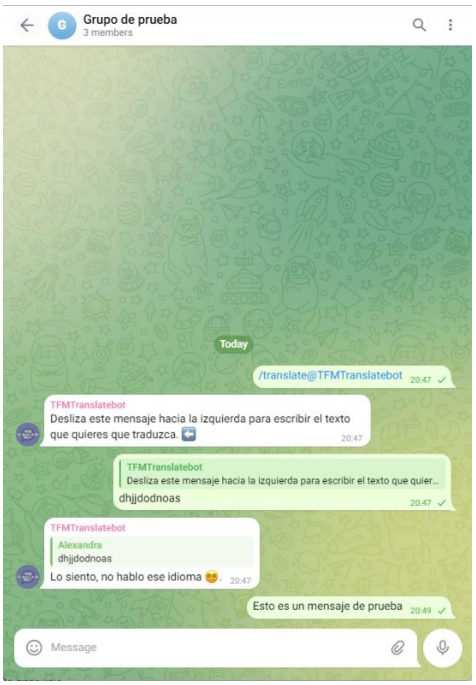

Figura 15: 9ª prueba en Telegram

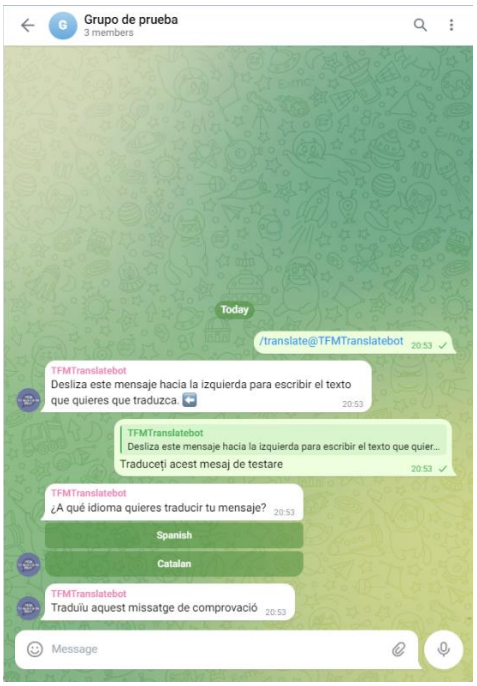

Figura 16: 10ª prueba en Telegram

```python
BOT_USERNAME = "TFMTranslatebot"
BOT_TOKEN = "6702118793:AAFmUDZyoU-2vTZoleuTfORVWm1uwKpRzAQ"
SERVER_IP = "localhost"
SERVER_PORT = 32772
APERTIUM_QUERY = f"http://{SERVER_IP}:{SERVER_PORT}"
USER_MESSAGE = ""

import telebot
import requests
import json
from iso639 import languages

mybot = telebot.TeleBot(BOT_TOKEN)
response = requests.get(f"{APERTIUM_QUERY}/listPairs")
response = json.loads(response.text)
LANG_PAIRS_LIST = response["responseData"]
LANG_PAIRS_DICT = {}
for item in LANG_PAIRS_LIST:
    source = item['sourceLanguage']
    target = item['targetLanguage']
    if source in LANG_PAIRS_DICT:
        LANG_PAIRS_DICT[source].append(target)
    else:
        LANG_PAIRS_DICT[source] = [target]

OTHER_LANGS = {
    "por_BR": "Brasilian Portuguese",
    "eng_US": "American English"
}

@mybot.message_handler(commands=['start', 'help'])
def start_cmd(message):
    if message.chat.type == "group":
        start_msg = "¡Hola!👋 Soy vuestro asistente de traducción automática 🤖." \
        "¿En qué puedo ayudaros? 🦾" \
        "Solo necesito que utilices el comando /translate para empezar a traducir 🌐."
    else:
        start_msg = "¡Hola!👋 Soy tu asistente de traducción automática 🤖." \
        "¿En qué puedo ayudarte? 🦾" \
        "Solo necesito que me envíes el mensaje que quieres traducir y sabré que idioma es 🧐."
    mybot.send_message(message.chat.id, start_msg)

@mybot.message_handler(commands=['translate'])
def translate_cmd_question(message):
    mybot.send_message(message.chat.id, "Desliza este mensaje hacia la izquierda" \
        " para escribir el texto que quieres que traduzca. ⬅️")
    mybot.register_next_step_handler(message, translate_cmd_answer)

def translate_cmd_answer(message):
    message.chat.type = 'command'
    menu(message)

@mybot.message_handler(func=lambda msg: True)
def menu(message):
    if message.chat.type == 'group':
        return
    langcode = detect_lang(message.text)
    if langcode is None:
        mybot.reply_to(message, "Lo siento, no hablo ese idioma 🤓.")
    else:
        target_list = LANG_PAIRS_DICT[langcode]
        menu = telebot.types.InlineKeyboardMarkup(row_width=2)
        for item in target_list:
            try:
                langname = languages.get(part3=item).name
            except Exception:
                if item in OTHER_LANGS.keys():
                    langname = OTHER_LANGS[item]
                else:
                    continue
            element = telebot.types.InlineKeyboardButton(langname, callback_data=f"{langcode}|{item}")
            menu.add(element)
        global USER_MESSAGE
        USER_MESSAGE = message.text
        mybot.send_message(message.chat.id, "¿A qué idioma quieres traducir tu mensaje?", reply_markup=menu)
```

```python
@mybot.callback_query_handler(func=lambda call: True)
def reply(answer):
    chat_id = answer.message.chat.id
    lang_pair = answer.data.split('|')
    translated_message = translate(lang_pair[0],lang_pair[1], USER_MESSAGE)
    mybot.send_message(chat_id, translated_message)

def detect_lang(message):
    myquery = f"{APERTIUM_QUERY}/identifyLang?q={message}"
    identity = requests.get(myquery)
    identity = json.loads(identity.text)
    selected_lang = max(identity, key=identity.get)
    selected_lang_prob = identity[selected_lang]
    if selected_lang_prob < 0.1:
        return None
    else:
        return selected_lang

def translate(source, target, message):
    myquery = f"{APERTIUM_QUERY}/translate?langpair={source}|{target}&q={message}"
    response = requests.get(myquery)
    response = json.loads(response.text)
    translation = response["responseData"]["translatedText"]
    return translation

mybot.infinity_polling()
```

Figura 17: Código de la 5ª propuesta

Previamente, se detectó que había dos lenguas que no aparecían por tratarse de lenguas con más de tres caracteres y, por tanto, el método *languages* no las reconocía. Por ende, en primer lugar, para subsanar este problema, se crea un diccionario que contiene el código de idioma como clave y su nombre como valor.

```python
OTHER_LANGS = {
    "por_BR": "Brasilian Portuguese",
    "eng_US": "American English"
}
```

El mensaje automático de inicio no podía reutilizarse para los grupos, por lo que se añadió una condición *if* que verifica si el tipo de chat es un grupo. Si es así, se envía un mensaje específico para grupos; de lo contrario, se ejecuta el bloque bajo *else*, que contiene el mensaje ya configurado para chats privados.

```python
@mybot.message_handler(commands=['start', 'help'])
def start_cmd(message):
    if message.chat.type == "group":
        start_msg = "¡Hola!👋 Soy vuestro asistente de traducción automática 🤖." \
        "¿En qué puedo ayudaros? 🦾" \
        "Solo necesito que utilices el comando /translate para empezar a traducir 🌐."
    else:
        start_msg = "¡Hola!👋 Soy tu asistente de traducción automática 🤖." \
            "¿En qué puedo ayudarte? 🦾 " \
            "Solo necesito que me envíes el mensaje que quieres traducir y sabré que idioma es 🧐."
    mybot.send_message(message.chat.id, start_msg)
```

A continuación, se define un decorador que indica que la función *translate_cmd_question* solo manejará los mensajes que contengan el nuevo comando, */translate*. Así, al recibir el

comando, se envía el mensaje automático definido con las indicaciones que seguir por el usuario. Después, se ejecuta la función *register_next_step_handler* que indica que el siguiente paso es llamar a la función *translate_cmd_answer.*

```
49  @mybot.message_handler(commands=['translate'])
50  def translate_cmd_question(message):
51      mybot.send_message(message.chat.id,
52          "Desliza este mensaje hacia la izquierda" \
53          "para escribir el texto que quieres que traduzca. ⬅️")
54      mybot.register_next_step_handler(message, translate_cmd_answer)
```

Por tanto, luego, se define la función *translate_cmd_answer.* Esta se encarga de manejar los mensajes que el usuario envía tras seguir las instrucciones del mensaje automático del comando */translate*. Luego, se cambia el tipo de chat a *command*, el propósito se explicará más adelante. Posteriormente, se llama a la función *menu* con el mensaje como argumento.

```
55  def translate_cmd_answer(message):
56      message.chat.type = 'command'
57      menu(message)
```

Para poder introducir el menú en los grupos, se han hecho cambios en la función. Se introduce una condición *if* que verifica si el tipo de chat es grupo y, si es así, el bot no lleva a cabo ninguna acción. El objetivo es que el bot no interrumpa ninguna conversación entre usuarios del grupo y solo se ejecute si se utiliza el comando */translate*, que al cambiar el tipo de chat sí que permite la ejecución del resto del código de la función.

```
59  @mybot.message_handler(func=lambda msg: True)
60  def menu(message):
61      if message.chat.type == 'group':
62          return
```

Además, se ha configurado para que, si ocurre una excepción, en vez de continuar si *languages* no consigue obtener el nombre del idioma, se verifique si el nombre del idioma se encuentra en las claves del diccionario *OTHER_LANGS*. Si está presente, se asigna el nombre del idioma como el valor del diccionario; de lo contrario, el código continúa.

```
64              for item in target_list:
65                  try:
66                      langname = languages.get(part3=item).name
67                  except Exception:
68                      if item in OTHER_LANGS.keys():
69                          langname = OTHER_LANGS[item]
70                      else:
71                          continue
```

3. Cavilaciones finales

Para concluir este arduo proceso de programación, se ha considerado imperante la inclusión de dos diagramas de secuencias, las Figuras 18 y 19. Estos ilustran de forma clara y concisa el flujo y la interacción entre el usuario, el bot y el servidor de Apertium, tanto en chats privados como en grupos. Los diagramas facilitan la comprensión del comportamiento del bot en diferentes escenarios, resaltando los pasos claves en la ejecución y la lógica detrás de cada función implementada.

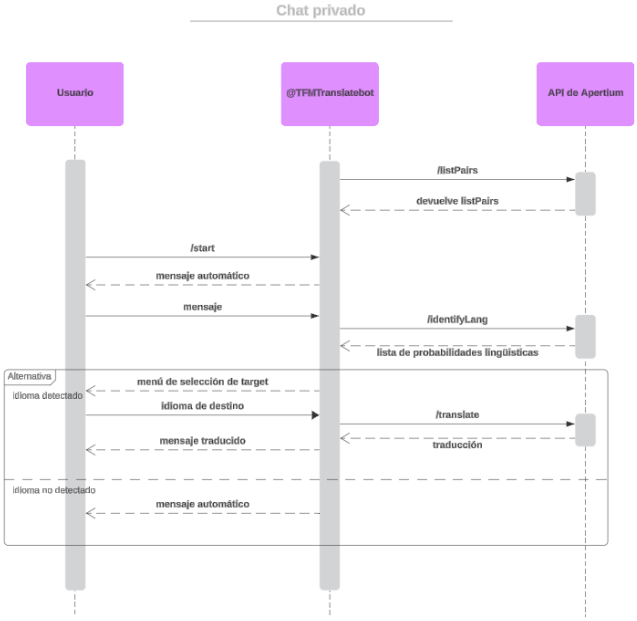

Figura 18: Diagrama de secuencias en chat privado

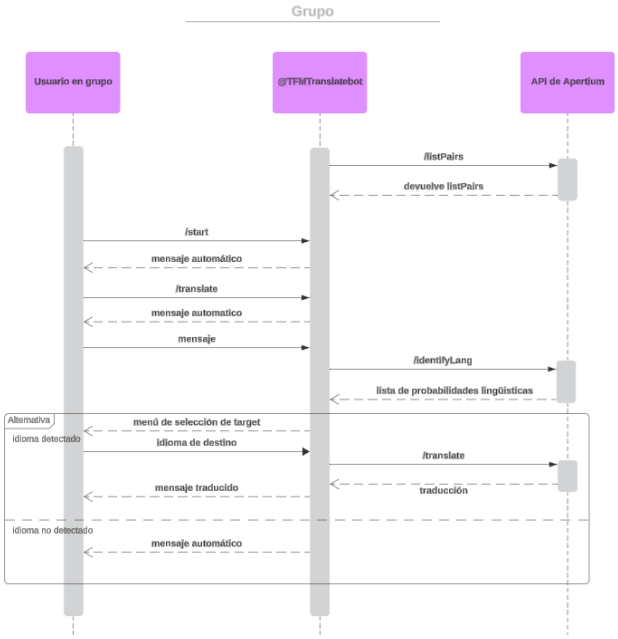

Figura 19: Diagrama de secuencias en grupos

A lo largo de los puntos anteriores se ha evidenciado la viabilidad y el funcionamiento del bot, cumpliendo con los objetivos planteados al inicio del proyecto. El bot ha demostrado ser una herramienta efectiva para la traducción automática de mensajes en Telegram, ofreciendo una solución práctica para la comunicación multilingüe. Sin embargo, el proyecto aún tiene un potencial significativo sin explotar, principalmente debido a limitaciones de tiempo y recursos.

Por ende, existen varias áreas de mejoras que puede ser una oportunidad para investigaciones futuras que podrían abordar mejoras en la funcionalidad del bot. A continuación, se expondrán algunas ideas que podrían explorarse en un futuro.

Una de las principales mejoras sería la implementación de modelos de traducción basados en redes neuronales. Estos modelos, conocidos por su precisión y capacidad de aprendizaje, podrían aumentar notablemente la calidad y exactitud de las traducciones, superando las limitaciones de los métodos actuales.

Además, la ampliación de las capacidades del bot para incluir la traducción de imágenes y audios sería un avance significativo. La traducción de imágenes requeriría la integración de tecnología de reconocimiento óptico de caracteres (OCR), mientras que la traducción de audios necesitaría la implementación de sistemas de reconocimiento de voz. Estas funcionalidades permitirían a los usuarios traducir una gama más amplia de contenido, haciendo del bot una herramienta aún más versátil y útil.

Otra posible mejora sería la capacidad de traducir mensajes antes de enviarlos. Esta funcionalidad permitiría a los usuarios previsualizar la traducción de sus mensajes, garantizando que la comunicación sea clara y precisa antes de ser compartida. Esto no solo mejoraría la experiencia del usuario, sino que también podría reducir la cantidad de mensajes incorrectos o malinterpretados.

En conclusión, aunque el proyecto ha alcanzado sus objetivos iniciales con éxito, hay varias oportunidades para continuar su desarrollo y mejora. Estas futuras investigaciones y desarrollos no solo incrementarían la utilidad y refinamiento del bot, sino que también contribuirían a la evolución de las herramientas de traducción automática, beneficiando a una audiencia global más amplia.

V. Conclusiones

En última instancia, como ya adelantábamos en el apartado anterior, es posible afirmar que el objetivo inicial de programar y ejecutar un bot funcional para Telegram dedicado a la traducción automática de mensajes se ha cumplido. Este proyecto se logra abordar y concebir gracias a un interés activo de poner en práctica los conocimientos básicos de programación que se brindan en el máster y una intención tenaz de contribuir a la mejora y avance de la comunicación digital junto a la traducción automática y las nuevas tecnologías.

Luego, es relevante destacar la utilidad y funcionalidad de la herramienta desde los primeros estadios del desarrollo, brindando a los usuarios la capacidad de tener un traductor capaz al alcance de la mano. Atendiendo a las posibles aplicaciones de la herramienta desarrollada, se considera su potencial para ser aplicada en diversos ámbitos, facilitando la interacción entre personas de diferentes idiomas y promoviendo el entendimiento intercultural. Este bot puede ser especialmente útil en sectores como la educación, el turismo, y las comunicaciones internacionales, donde la barrera del idioma es un desafío constante.

A lo largo del desarrollo, se enfrentaron varios desafíos técnicos, como la integración de la API de Apertium y la detección automática de idiomas. Estos desafíos fueron superados mediante una cuidadosa planificación y pruebas exhaustivas. Además, el bot fue sometido a pruebas rigurosas para asegurar su funcionalidad y precisión. Así, la implementación de algoritmos más avanzados ha permitido la mejora y adaptación del bot con el fin de ampliar la variedad de idiomas, la precisión de detección y la optimización de la interfaz de usuario, consiguiendo garantizar una experiencia cómoda e intuitiva y unos resultados adecuados.

Asimismo, si bien la utilidad práctica es imperante, es relevante destacar que el desarrollo e implementación del bot proporcionan una oportunidad única de aprendizaje personal y académico. Los diferentes procesos que se han llevado a cabo, desde el diseño y la programación hasta el testeo y la evaluación, han requerido una documentación compleja y exhaustiva que han resultado en una comprensión y asimilación de conocimientos amplios y útiles.

Así, si bien se han considerado las fortalezas y logros del proyecto, también se han discutido las posibles áreas de mejora y se ha especulado sobre cómo llevar ciertas implementaciones a cabo. Estos avances no solo incrementarían la utilidad y refinamiento del bot, sino que también contribuirían a la evolución de la implementación de este tipo de herramientas de traducción automática para una audiencia más amplia.

Este proyecto ha sido una oportunidad única para aplicar y expandir mis conocimientos en programación y traducción automática. A través del diseño, implementación y evaluación del bot, he adquirido una comprensión profunda de estos campos, lo que me prepara para futuros desafíos y proyectos en el ámbito del aprendizaje automático y la tecnología de la información.

Por tanto, se puede concluir que el trabajo ha resultado un proyecto soberbio por su capacidad de otorgar valiosos conocimientos de los ámbitos de la traducción y la programación. Además del positivo impacto académico que brinda, la posibilidad de su uso posterior ofrece una herramienta útil y completa al mismo tiempo que contribuye a la accesibilidad y la comunicación en un mundo globalizado.

VI. Bibliografía

Adamopoulou, E. y Moussiades, L. (2020). Chatbots: History, technology, and applications. *Machine Learning with applications, 2*(100006), pp. 1-18. https://doi.org/10.1016/j.mlwa.2020.100006

Armentano Oller, C., Corbí Bellot, A. M., Forcada, M. L., Ginestí-Rosell, M., Montava Belda, M. A., Ortiz Rojas, S., Pérez-Ortiz, J. A., Ramírez Sánchez, G. y Sánchez-Martínez, F. (2007). Apertium, una plataforma de código abierto para el desarrollo de sistemas de traducción automática. En J. R. Rodríguez Galván y M. Palomo Suarte (coords.), *Proceedings of the FLOSS International Conference 2007* (pp. 1-16). Servicio de Publicaciones de la Universidad de Cádiz.

Babhulgaonkar, A R. y Bharad, S. V. (2017). Statistical machine Translation. *2017 1st International Conference on Intelligent Systems and Information Management (ICISIM),* pp. 62-67. https://doi.org/10.1109/icisim.2017.8122149

Bar-Hillel, Y. (1952). The present state of research on mechanical translation. *American Documentation*, *2*(4), pp. 229-237. https://doi.org/10.1002/asi.5090020408

Bhattacharyya, P. (2015). *Machine translation*. CRC Press.

Booth, A. D. (1958). The history and recent progress of machine translation en Smith, H y Booth, A.D. (Ed.) *Aspects of translation* (pp. 89-104). Secker and Warburg.

Docker. (s.f.). *Docker Documentation*. Docker. Recuperado el 8 de mayo de 2024 de https://docs.docker.com/get-started/overview/

Forcada, M. L. (2009). Apertium: traducció automàtica de codi obert per a les llengües romàniques. *Linguamática, 1*(1), pp. 13-23.

Forcada, M. L. (2015). Open-source machine translation technology. En S. Chan (Ed.), *Routledge Encyclopedia of Translation Technology* (pp. 152-166). Routledge.

Forcada, M. L. (2017). Making sense of neural machine translation. Translation Spaces, 6(2), 291–309. https://doi.org/10.1075/ts.6.2.06for

Forcada, M. L., Ginestí-Rosell, M., Nordfalk, J., O'Regan, J., Ortiz-Rojas, S., Pérez-Ortiz, J. A., Sánchez-Martínez, F., Ramírez-Sánche, G. y Tyers, F. M. (2011). Apertium: a free/open-source platform for rule-based machine translation. *Machine Translation, 25*(2), pp. 127–144. https://doi.org/10.1007/s10590-011-9090-0

Ginestí, M. y Forcada, M. L. (2009). La traducció automàtica en la pràctica: aplicacions, dificultats i estratègies de desenvolupament. *Caplletra. Revista Internacional de Filologia*, (46), pp. 43-60.

Harper Collins. (s. f.). Machine Translation. En el *Diccionario Collins*. Recuperado el 13 de marzo de 2024 de https://www.collinsdictionary.com/dictionary/english/machine-translation

Hasal, M., Nowaková, J., Saghair, K. A., Abdulla, H., Snášel, V. y Ogiela L. (2021). Chatbots: Security, privacy, data protection, and social aspects. *Concurrency and Computation: Practice and Experience, 33*(19), pp. 1-13. https://doi.org/10.1002/cpe.6426

Hendy, A., Abdelrehim, M., Sharaf, A., Raunak, V., Gabr, M., Matsushita, H., Kim, Y. J., Afify, M. y Awadalla, H. H. (2023). How Good Are GPT Models at Machine Translation? A Comprehensive Evaluation. https://doi.org/10.48550/arXiv.2302.09210

Hieber, F. y Domhan, T. (20 de julio de 2017). Train Neural Machine Translation Models with Sockeye. *AWS Machine Learning Blog.* https://aws.amazon.com/es/blogs/machine-learning/train-neural-machine-translation-models-with-sockeye/

Hutchins, J. (1999). The historical development of machine translation. *Submission for the Degree of Doctor of Philosophy by Publication at the University of East Anglia.*

Hutchins, J. (2007). Machine translation: A concise history. *Computer aided translation: Theory and practice*, *13*(29-70), p. 11.

Hutchins, W. J. (2003). The development and use of machine translation systems and computer-based translation tools. *International Journal of Translation,* 15(1), pp. 5-26.

Hutchins, W. J., y Somers, H. L. (1992). *An Introduction fo Machine Translation*. Academic Press.

Jiang, K y Lu, X. (2021). Integrating Machine Translation with Human Translation in the Age of Artificial Intelligence: Challenges and Opportunities. En M. Atiquzzaman, N. Yen y Z. Xu (Eds*.), Big Data Analytics for Cyber-Physical System in Smart City. BDCPS 2020. Advances in Intelligent Systems and Computing* (Vol. 1303, pp. 1397-1405). Springer. https://doi.org/10.1007/978-981-33-4572-0_202

Jiao, W., Wang, W., Huang, J. T., Wang, X., Shi, S. y Tu, Z. (2023). Is ChatGPT a good translator? Yes with GPT-4 as the engine. https://doi.org/10.48550/arXiv.2301.08745

Kenny, D. (2022). *Machine translation for everyone: Empowering users in the age of artificial intelligence.* Language Science Press.

Khan, R y Das, A. (2017). *Build Better Chatbots: A Complete Guide to Getting Started with Chatbots.* Apress Berkeley, CA. https://doi.org/10.1007/978-1-4842-3111-1

Khanna, T., Washington, J.N., Tyers, F.M., Bayatlı, S., D. G., Pirinen, T. A., Tang, I. y Alòs i Font, H. (2021). Recent advances in Apertium, a free/open-source rule-based machine translation platform for low-resource languages. *Machine Translation 35*, pp. 475–502. https://doi.org/10.1007/s10590-021-09260-6

Kunst, J. R. y Bierwiaczonek, K. (2023). Utilizing AI questionnaire translations in cross-cultural and intercultural research: Insights and recommendations*. International Journal of Intercultural Relations, 97*(101888), pp. 1-11. https://doi.org/10.31234/osf.io/sxcyk

Liu, Q. y Zhang, Z. (2015). Machine Translation: General. En S. Chan (Ed.), *The Routledge Encyclopedia of Translation Technology* (pp. 105-119). Routledge.

López, A. (2008). Statistical machine translation. *ACM Computing Surveys, 40*(3), pp. 1-49. https://doi.org/10.1145/1380584.1380586

Lyu, C., Xu, J. y Wang, L. (2024). New trends in machine translation using large language models: case examples with ChatGPT.

Maher, S., Kayte, S., & Nimbhore, S. (2020). Chatbots & its techniques using AI: an review. *International journal for research in applied science and engineering technology, 8*(12), pp. 503-508. https://doi.org/10.22214/ijraset.2020.32537

Microsoft Translator. (15 de noviembre de 2016). Microsoft Translator launching Neural Network based translations for all its speech languages. *Microsoft Translator Blog.* https://www.microsoft.com/en-us/translator/blog/2016/11/15/microsoft-translator-launching-neural-network-based-translations-for-all-its-speech-languages/

Mondal, S. K., Zhang, H., Kabir, H. D., Ni, K., y Dai, H. N. (2023). Machine translation and its evaluation: a study. *Artificial Intelligence Review*, *56*(9), pp. 10137-10226. https://doi.org/10.1007/s10462-023-10423-5

Murphy, H. (11 de marzo de 2024). *Telegram hits 900mn users and nears profitability as founder considers IPO*. Financial Times. https://www.ft.com/content/8d6ceb0d-4cdb-4165-bdfa-4b95b3e07b2a

Parra Escartín, P. (2018). ¿Cómo ha evolucionado la traducción automática en los últimos años? *La Linterna del Traductor,* (16), pp. 20-28.

Pérez-Ortiz, J. A., Forcada, M. L. y Sánchez-Martínez, F. (2022). How neural machine translation works. En D. Kenny (Ed.), *Machine translation for everyone: Empowering users in the age of artificial intelligence* (pp. 141-164). Language Science Press.

Poibeau, T. (2017). *Machine translation*. MIT Press.

Quah, C.K. (2006). Machine Translation Systems en Quah, C.K. (Ed.), *Translation and Technology* (pp. 57-92). Palgrave Macmillan. https://doi.org/10.1057/9780230287105_4

Sánchez Ramos, M. D. M., & Rico Pérez, C. (2020). *Traducción automática. Conceptos clave, procesos de evaluación y técnicas de posedición*. Comares.

Sheeren, S.A., Elsayed, A.A., Hassan, Y.F. y Abdou, M.A. (2021). Neural machine translation: past, present, and future. *Neural Computing and Applications, 33*, pp. 15919–15931. https://doi.org/10.1007/s00521-021-06268-0

Sidorov, A. (3 de agosto de 2017). Transitioning entirely to neural machine translation. *Engineering at Meta.* https://engineering.fb.com/2017/08/03/ml-applications/transitioning-entirely-to-neural-machine-translation/

Singh, S., Xu, M., Patros, P., Wu, H., Kaur, R., Kaur, K., Fuller, S., Singh, M., Arora, P., Parlikad, A. K., Stankovski, V., Abraham, A., Ghosh, S. K., Lutfiyya, H., Kanhere, S. S., Bahsoon, R., rana, O., Dustdar, S., Sakellariou, R., Uhlig, S. y Buyya, R. (2024). Transformative effects of ChatGPT on modern education: Emerging Era of AI Chatbots. *Internet of Things and Cyber-Physical Systems, 4*, pp. 19-23. https://doi.org/10.1016/j.iotcps.2023.06.002

Stein, D. (2013). Machine Translation: Past, Present and Future. Translation: Computation, Corpora. *Cognition, 3*(1), pp. 5-13.

Suta, P., Lan, X., Wu, B., Mongkolnam, P., y Chan, J. H. (2020). An overview of machine learning in chatbots. *International Journal of Mechanical Engineering and Robotics Research, 9*(4), 502-510. https://doi.org/10.18178/ijmerr.9.4.502-510

Taye, M. M. (2023). Understanding of machine learning with deep learning: architectures, workflow, applications and future directions. *Computers, 12*(5), pp. 91. https://doi.org/10.3390/computers12050091

Telegram. (16 de abril de 2022). *Notification Sounds, Bot Revolution and More*. Telegram Blog. https://telegram.org/blog/notifications-bots?ln=r

Telegram. (24 de junio de 2015). *Telegram Bot Platform*. Telegram Blog. https://telegram.org/blog/bot-revolution

Telegram. (s.f.-a). *Bots: An introduction for developers*. Telegram Core. Recuperado el 2 de mayo de 2024 de https://core.telegram.org/bots

Telegram. (s.f.-b). *Telegram Bot Features*. Telegram Core. Recuperado el 2 de mayo de 2024 de https://core.telegram.org/bots/features

Trujillo, A. (1999). *Translation Engines: Techniques for Machine Translation*. Springer London. https://doi.org/10.1007/978-1-4471-0587-9

Wang, H., Wu, H., He, Z., Huang, L., y Church, K. W. (2022a). Progress in machine translation. *Engineering*, *18*, pp. 143-153. https://doi.org/10.1016/j.eng.2021.03.023

Wang, W., Jiao, W., Hao, Y., Wang, X., Shi, S., Tu, Z., y Lyu, M. (2022b). Understanding and Improving Sequence-to-Sequence Pretraining for Neural Machine Translation. *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, 1*, pp. 2591-2600. https://doi.org/10.48550/arXiv.2203.08442

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. https://doi.org/10.48550/arXiv.1609.08144

Yu, S. y Bai, X. (2015). Rule-based machine translation. En S. Chan (Ed.), The Routledge *Encyclopedia of Translation Technology* (pp. 186–200). Routledge.

Zhao, Y., Zhang, J. y Zong, C. (2023) Transformer: A General Framework from Machine Translation to Others. *Machine Intelligence Research, 20*, pp. 514-538. https://doi.org/10.1007/s11633-022-1393-5