Sistema de Automatización y Procesamiento Inteligente para la Traducción y Maquetación de Documentos Bilingües: Un Enfoque basado en Inteligencia Artificial.

TRABAJO DE FIN DE GRADO

GRADO EN ESTADÍSTICA EMPRESARIAL

FACULTAD DE CIENCIAS SOCIALES Y JURÍDICAS DE ELCHE

UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

Curso académico 2024 - 2025

Autor: Raúl Fuster Martínez

Tutor: Fernando Borrás Rocher

Resumen

En este Trabajo de Fin de Grado se presenta el desarrollo de un sistema que automatiza la traducción y maquetación de documentos, concretamente las Resoluciones Rectorales de la Universidad Miguel Hernández (UMH). Se ha logrado garantizar su formato bilingüe en valenciano y castellano mediante dos enfoques diferentes. Además de optimizar el proceso de traducción, el sistema mantiene la coherencia visual y estructural de los documentos oficiales.

El primer método emplea inteligencia artificial a través de la API de ChatGPT, mientras que el segundo utiliza el traductor Apertium mediante Selenium para realizar la traducción de forma automatizada.

Finalmente, se comparan ambas soluciones para evaluar su precisión, eficiencia y viabilidad dentro del contexto académico y administrativo de la UMH.

Palabras clave

Automatización, traducción automática, inteligencia artificial, apertium

Introducción

La idea de este trabajo inició luego de cursar la asignatura de Mejora de Procesos, impartida por María Asunción Martínez Mayoral, profesora en el Grado de Estadística Empresarial. Nos comentó en repetidas ocasiones las posibilidades que las inteligencias artificiales y la programación tienen en nuestro ámbito, y me picó la curiosidad. Le propuse un tema para el trabajo sobre una automatización, y ella me derivó al que, finalmente, es mi tutor para abordar este proyecto, Fernando Borrás Rocher. Nos reunimos y él me propuso el tema que terminó siendo el seleccionado.

Como todos sabemos, vivimos en un mundo cada vez más digitalizado, y la automatización de procesos es ya una herramienta clave para la optimización del tiempo, de recursos y de tareas en muchas áreas diferentes.

En el sector administrativo, la traducción de documentos oficiales es una tarea fundamental. Además, en un territorio como es la Comunidad Valenciana, la cual

tiene dos idiomas oficiales como son el castellano y el valenciano, lo es aún más si cabe.

El Boletín Oficial de la Universidad Miguel Hernández (BOUMH) publica resoluciones rectorales diariamente, a veces incluso más de una al día, y son presentadas en formato bilingüe, con el texto en castellano y valenciano distribuido de forma simétrica en cada página. Al haber tal cantidad de resoluciones diarias, desarrollar un sistema de automatización que es capaz de traducir y maquetar todas estas resoluciones de forma rápida, eficiente y sencilla, es una manera de agilizar el trabajo y poder centrarse en otras cuestiones con más dificultad o carga de trabajo.

Con este trabajo no se pretende únicamente optimizar un proceso administrativo de la universidad, si no también se quiere demostrar que las nuevas tecnologías pueden integrarse perfectamente en el ámbito público, aumentando la eficiencia y calidad del trabajo.

El sistema desarrollado tiene el potencial de ser escalable y adaptable a otros contextos y entidades que necesiten manejar documentos bilingües de manera eficiente.

Objetivos

El objetivo general de este Trabajo de Fin de Grado es desarrollar e implementar un sistema automatizado que permita traducir y maquetar documentos oficiales de la Universidad Miguel Hernández (UMH) de forma rápida y eficiente, garantizando un formato bilingüe en castellano y valenciano. Para lograrlo, se han explorado dos enfoques distintos: uno basado en la integración de la API de ChatGPT y otro utilizando el traductor Apertium a través de Selenium.

Objetivos Específicos

- **Automatización de la traducción y maquetación:** Desarrollar un sistema que automatice el proceso de traducción de documentos oficiales y la maquetación de dichos textos en formato

bilingüe, asegurando que la estructura y el diseño institucional se mantengan inalterados.

- **Implementación de dos métodos de traducción:** Integrar y poner en funcionamiento dos soluciones de traducción automática:
    - **Método Apertium vía Selenium:** Configurar el entorno y desarrollar el código que permita traducir el contenido mediante Apertium, automatizando la interacción con la web mediante Selenium.
    - **Método basado en la API de ChatGPT:** Desarrollar e implementar el código necesario para traducir textos a través de la API de ChatGPT, aprovechando las capacidades de la inteligencia artificial para generar traducciones naturales y precisas.
- **Evaluación comparativa:** Comparar ambos métodos en términos de rendimiento, precisión y eficiencia, analizando las ventajas y limitaciones de cada enfoque para determinar su viabilidad en el contexto administrativo y académico de la UMH.
- **Propuesta de mejoras y futuras líneas de trabajo:** Identificar posibles mejoras en el sistema desarrollado y proponer futuras líneas de investigación o implementación que permitan optimizar aún más el proceso de traducción y maquetación de documentos oficiales.

Información disponible

Para llevar a cabo el desarrollo de este Trabajo de Fin de Grado, se ha recopilado y utilizado información tanto técnica como académica, con el fin de comprender los procesos involucrados en la automatización de la traducción y maquetación de documentos oficiales en la Universidad Miguel Hernández (UMH). A continuación, se detallan las fuentes y tipos de información disponibles:

1. **Documentación técnica sobre Apertium y Selenium:**
    - **Apertium:** Apertium es un sistema de traducción automática de código abierto que soporta múltiples combinaciones lingüísticas, incluido el par de idiomas castellano-valenciano. La información técnica sobre Apertium, como su funcionamiento y su implementación en diferentes lenguajes de programación, ha sido esencial para integrar el traductor en el sistema automatizado.
    - **Selenium:** Selenium es una herramienta que permite automatizar la interacción con aplicaciones web, siendo útil para interactuar con el traductor Apertium a través de su interfaz web. Se ha consultado la documentación de Selenium para configurar y automatizar el navegador, garantizando que el sistema funcione de manera eficiente.
2. **API de OpenAI (ChatGPT):**
    - La API de OpenAI, utilizada para implementar el sistema de traducción automática basado en inteligencia artificial, proporciona una interfaz para acceder a modelos de lenguaje avanzados, como GPT-4. La información sobre cómo realizar las solicitudes a la API, manejar las respuestas y adaptar los resultados para que sean adecuados al contexto de traducción, se obtuvo a partir de la documentación oficial de OpenAI.
3. **Especificaciones de los documentos oficiales de la UMH:**
    - **Plantillas y ejemplos:** Se ha tenido acceso a ejemplos de documentos oficiales previamente publicados, que sirven como referencia para la maquetación y disposición de los textos en castellano y valenciano, asegurando la correcta distribución del contenido.

4. **Recursos de formación y investigación:**
    - Mucha de la información utilizada para realizar este proyecto ha sido extraída del curso que imparte el tutor Fernando Borrás Rocher, llamado “AutomatizandoIA”, dentro del programa “IA4Legos”.
    - Se ha consultado libros y artículos académicos relacionados con la traducción automática, la automatización de procesos administrativos y el uso de herramientas como Apertium y Selenium en la industria.
5. **Herramientas y entornos de desarrollo:**
    - Se ha utilizado un entorno de desarrollo basado en Python, con bibliotecas como **Selenium** y **python-docx**, que facilitan la interacción con aplicaciones web y la manipulación de documentos Word, respectivamente. La documentación oficial de estas herramientas ha sido clave para asegurar una correcta integración y funcionamiento del sistema.
    - También se ha hecho uso de Google Colab para ejecutar el código en un entorno controlado y disponer de recursos de computación adecuados para las pruebas y validaciones del sistema desarrollado.

6. Metodología

En este apartado se describe la metodología seguida para el desarrollo del sistema automatizado de traducción y maquetación de documentos bilingües en el contexto de las resoluciones rectorales del Boletín Oficial de la Universidad Miguel Hernández (BOUMH). La metodología abarca todo el proceso desde el análisis de los requisitos y especificaciones hasta la implementación, pruebas y evaluación del sistema. El enfoque adoptado en este proyecto ha sido el de crear una solución integral que permita traducir y presentar de manera coherente y eficiente los documentos de las resoluciones rectorales en dos lenguas oficiales: castellano y valenciano. Para ello, se desarrollaron y evaluaron diferentes

enfoques y tecnologías, y se adoptó una metodología iterativa y centrada en la calidad.

El desarrollo de este sistema sigue una serie de pasos detallados que aseguran que los documentos sean traducidos y maquetados adecuadamente, respetando tanto la precisión lingüística como la correcta disposición del contenido en el formato deseado. A continuación, se describen los pasos clave en la metodología de desarrollo del sistema.

6.1 Revisión de requisitos y especificaciones

El primer paso del proyecto consistió en realizar un análisis detallado de los requisitos y las especificaciones del Boletín Oficial de la Universidad Miguel Hernández (BOUMH), que se caracteriza por publicar resoluciones rectorales en formato bilingüe (castellano y valenciano). Este análisis inicial fue crucial para sentar las bases del proyecto, ya que permitió identificar las necesidades específicas tanto en el ámbito de la traducción automática como en el de la maquetación adecuada de los documentos.

Análisis del formato y estructura de los documentos

Una de las principales tareas fue estudiar cómo se presentaban las resoluciones rectorales en el BOUMH. El formato de estos documentos, que se caracteriza por una presentación bilingüe y una distribución simétrica de los textos, era fundamental para asegurar que el sistema automatizado cumpliera con las expectativas de presentación y precisión. Durante este análisis, se identificaron los siguientes aspectos clave:

1. **Distribución simétrica de los textos**: Los textos debían ser presentados en dos columnas, con el castellano en una columna y el valenciano en la otra, de forma que ambas lenguas se mantuvieran perfectamente alineadas. Esto era esencial para garantizar la legibilidad y comprensión del contenido en ambas lenguas.
2. **Incorporación de elementos gráficos**: El BOUMH suele incluir ciertos elementos gráficos, como el logotipo de la universidad y los encabezados institucionales, que también debían ser preservados y ubicados adecuadamente en el documento final.

3. **Formato específico de los títulos y subtítulos**: Los documentos seguían un formato específico para los títulos y subtítulos, con diferentes tamaños y estilos de fuente. Era necesario asegurar que la maquetación automatizada respetara estas convenciones de estilo.

Investigación de herramientas tecnológicas

Una vez comprendida la estructura de los documentos, el siguiente paso fue la investigación de las herramientas tecnológicas más adecuadas para implementar la traducción automática y la maquetación del documento. Para esto, se realizaron las siguientes acciones:

1. **Selección de soluciones de traducción automática**:
    - **Apertium**: Se optó por Apertium para realizar la traducción entre castellano y valenciano, dado que esta herramienta es ampliamente utilizada para lenguas cercanas y ofrece traducción automática en el par lingüístico castellano-valenciano. Apertium es una solución basada en reglas, lo que garantiza una alta precisión para los textos administrativos y legales, como las resoluciones rectorales.
    - **API de ChatGPT**: Como una alternativa complementaria, se investigó el uso de la API de ChatGPT para ofrecer traducciones más fluídas y contextuales. Esta solución basada en inteligencia artificial fue utilizada para garantizar que el texto final no solo fuera preciso, sino también coherente y natural en cuanto a la redacción.
2. **Selección de herramientas para la maquetación**:
    - **python-docx**: Esta biblioteca fue elegida para manipular los documentos en formato Word. Permite crear, modificar y dar formato a documentos de manera eficiente, lo que facilitó la tarea de crear documentos bilingües con la disposición adecuada de los textos.
    - **Selenium**: Esta herramienta fue utilizada para automatizar la interacción con la interfaz web de Apertium, facilitando el proceso

de traducción directa desde el navegador y permitiendo la gestión del flujo de trabajo de manera completamente automatizada.

Definición de los requisitos funcionales

A partir de los análisis anteriores, se definieron los requisitos funcionales clave del sistema, que debían ser cumplidos en todas las etapas del desarrollo:

1. **Automatización del proceso de traducción**: El sistema debía ser capaz de automatizar todo el proceso de traducción de documentos, de forma que no fuera necesario realizar traducciones manuales para cada documento del BOUMH. La integración de Apertium y la API de ChatGPT permitió automatizar este proceso de manera eficiente.
2. **Integración de la traducción en un formato bilingüe simétrico**: Una vez realizada la traducción, el sistema debía presentar los documentos en un formato donde el texto en castellano y el texto en valenciano estuvieran dispuestos de manera simétrica y alineada. Esta tarea de maquetación debía ser también automatizada para que, al final del proceso, el documento estuviera listo para ser publicado en el BOUMH sin intervención manual.
3. **Validación de la calidad y coherencia del contenido traducido**: Asegurar la calidad de las traducciones era crucial. Esto implicaba que el sistema debía incorporar algún tipo de validación para comprobar que las traducciones generadas fueran precisas, coherentes y adecuadas al contexto administrativo y legal de las resoluciones rectorales. Además, debía poder verificar que la maquetación estuviera correctamente realizada, respetando las convenciones establecidas por la universidad para la publicación de documentos oficiales.

6.2 Diseño y desarrollo del sistema

El diseño y desarrollo del sistema para la automatización de la traducción y maquetación de los documentos bilingües del Boletín Oficial de la Universidad Miguel Hernández (BOUMH) se llevó a cabo en varias fases clave, cada una orientada a resolver un aspecto específico del proceso. Estas fases se

organizaron de manera secuencial para garantizar la optimización y eficiencia del sistema final. La planificación detallada de la arquitectura del sistema, el desarrollo de los módulos de traducción y la automatización del proceso de maquetación fueron los pilares sobre los cuales se construyó la solución final.

1. Arquitectura del sistema

La arquitectura del sistema fue diseñada con un enfoque modular, lo que permitió garantizar tanto su flexibilidad como su escalabilidad. Este enfoque modular facilita la actualización y mejora continua del sistema a medida que surgen nuevas tecnologías o requerimientos. Para lograr una implementación eficaz, se estableció un flujo de trabajo claro que facilitara la integración de los diferentes módulos del sistema, que son:

- **Módulo de traducción**: El componente encargado de traducir el contenido de los documentos del BOUMH de castellano a valenciano y viceversa. Este módulo utiliza tanto soluciones de traducción automática basadas en reglas como modelos de inteligencia artificial para mejorar la calidad de las traducciones.
- **Módulo de maquetación**: Encargado de estructurar los documentos traducidos en un formato adecuado y coherente con los requisitos institucionales del BOUMH. Este módulo asegura que los textos en castellano y valenciano estén distribuidos de manera simétrica y alineada.

Ambos módulos fueron diseñados para trabajar de forma autónoma, pero también para integrarse fácilmente dentro del flujo de trabajo global del sistema. De esta forma, se consiguió una solución que no solo cumpliera con los objetivos técnicos, sino que también fuera capaz de adaptarse a posibles cambios o ampliaciones en el futuro.

2. Módulo de traducción

El desarrollo del módulo de traducción fue uno de los aspectos más cruciales del sistema, ya que la calidad de la traducción era fundamental para la precisión y coherencia de los documentos oficiales. Para este módulo, se adoptaron dos enfoques diferentes:

- **Apertium vía Selenium**: Para integrar Apertium en el sistema, se utilizó Selenium, una herramienta que permite automatizar la interacción con interfaces web. Selenium facilitó la automatización del proceso de traducción al interactuar con la interfaz web de Apertium, enviando los textos a traducir y obteniendo las traducciones de manera eficiente y sin intervención manual. Esta solución fue especialmente útil para asegurar la consistencia y precisión de las traducciones, dada la naturaleza predefinida de las reglas lingüísticas en Apertium.
- **API de ChatGPT**: Para complementar Apertium y mejorar la fluidez de las traducciones, se integró la API de ChatGPT, que utiliza modelos avanzados de inteligencia artificial para generar traducciones contextuales y naturales. Este enfoque permitió manejar mejor los matices del lenguaje y las expresiones idiomáticas que podrían no ser bien tratadas por sistemas basados en reglas.

Ambos enfoques fueron diseñados para trabajar de manera independiente, lo que permitió compararlos en términos de eficiencia y precisión. Además, la combinación de estos métodos ofreció un balance adecuado entre la velocidad de traducción y la calidad contextual, asegurando una cobertura amplia de las necesidades del sistema.

4. Integración de los módulos

Los módulos de traducción y maquetación fueron integrados en un único flujo de trabajo automatizado. El sistema fue diseñado para permitir que un documento original en castellano fuera procesado de manera completamente automática, desde la traducción de su contenido hasta la maquetación final del documento bilingüe. La integración de ambos módulos permitió que, tras la traducción de los textos, el documento fuera automáticamente maquetado y preparado para su publicación en el BOUMH sin intervención manual.

Para garantizar que los resultados fueran satisfactorios, se realizaron varias pruebas con documentos de prueba para validar la efectividad del sistema. Estas pruebas incluyeron la evaluación de la precisión de las traducciones y la comprobación de que la maquetación de los textos en ambas lenguas cumpliera con los estándares de presentación requeridos por la universidad.

En resumen, el diseño y desarrollo del sistema se centró en la modularidad, la eficiencia y la facilidad de integración, asegurando que el proceso de traducción y maquetación de documentos fuera completamente automatizado y adecuado a los requerimientos institucionales del BOUMH.

6.3 Implementación de los métodos de traducción

La implementación de los métodos de traducción fue una parte esencial del sistema automatizado de traducción y maquetación. El desarrollo de los módulos de traducción con Apertium vía Selenium y con la API de ChatGPT se llevó a cabo con el objetivo de maximizar tanto la precisión como la fluidez de las traducciones, adaptándose a las particularidades de los documentos oficiales del BOUMH.

Desarrollo del módulo de traducción con Apertium vía Selenium

El primer módulo de traducción fue desarrollado utilizando **Apertium** como motor de traducción automática. Apertium es una plataforma basada en reglas que se especializa en las lenguas cercanas y tiene una excelente compatibilidad con la combinación de castellano y valenciano. Para interactuar con Apertium, se utilizó **Selenium**, una herramienta de automatización que permite controlar navegadores web de manera programática.

El proceso de implementación comenzó con la creación de un script que automatizara la carga del texto original en castellano en la interfaz web de Apertium y la obtención de la traducción al valenciano. Este enfoque fue elegido por su eficacia en la traducción de textos administrativos y legales, que son el tipo de contenido principal en el BOUMH. La automatización mediante Selenium permitió traducir documentos de manera eficiente sin necesidad de intervención manual, optimizando el proceso.

Desarrollo del módulo de traducción utilizando la API de ChatGPT

El segundo módulo de traducción fue desarrollado utilizando la **API de ChatGPT**, un modelo avanzado de inteligencia artificial que se especializa en el procesamiento de lenguaje natural. Este enfoque permitió obtener traducciones más contextuales, adaptadas a los matices y estilo del valenciano. A diferencia de Apertium, que se basa en un enfoque de traducción por reglas, la API de

ChatGPT genera traducciones más flexibles y naturales, lo que la convierte en una opción ideal para aquellos casos en los que se requiere una mayor adaptabilidad en el estilo del lenguaje.

El módulo de ChatGPT fue integrado con el sistema de manera que se pudiera elegir entre ambos métodos de traducción (Apertium o ChatGPT) dependiendo de las necesidades del documento y los resultados esperados. De esta forma, se aseguró que cada texto fuera traducido de la manera más adecuada según su contenido y contexto.

Integración de los dos métodos en el sistema para realizar comparativas

Una vez desarrollados ambos módulos, se procedió a integrarlos en un único sistema de traducción. Esto permitió realizar comparativas de rendimiento y calidad entre Apertium y ChatGPT, analizando cuál de los dos métodos producía mejores resultados en términos de precisión, fluidez y adaptabilidad. La integración también permitió automatizar la elección del método más adecuado según el tipo de texto o la naturaleza del documento.

Ambos módulos de traducción fueron evaluados mediante pruebas con documentos de prueba, para asegurar que el sistema produjera resultados de alta calidad tanto en términos de traducción como de maquetación.

6.4 Automatización de la maquetación de documentos

Una de las partes fundamentales de este proyecto fue la automatización de la maquetación de los documentos bilingües. El objetivo principal era garantizar una correcta distribución simétrica de los textos en castellano y valenciano, lo que implicaba que los textos de ambas lenguas debían estar alineados de manera precisa dentro del documento final.

Configuración de la maquetación del documento bilingüe

Para la correcta disposición de los textos en ambas lenguas, se diseñó una estructura de tabla que permitiera la colocación de las traducciones lado a lado. De este modo, se logró una distribución simétrica en la que el texto original en castellano ocupaba una columna, mientras que su traducción al valenciano se

colocaba en la columna adyacente. Además, la maquetación incluyó márgenes y un espaciado apropiado para asegurar que el texto fuera fácilmente legible y que ambas versiones del documento tuvieran una presentación coherente y uniforme.

Incorporación de elementos gráficos

Además de la disposición de los textos, el sistema permitió la incorporación de elementos gráficos, tales como los logotipos institucionales de la Universidad Miguel Hernández, que deben aparecer en todos los boletines oficiales. Para ello, se integraron las herramientas adecuadas para insertar imágenes en el documento de manera automatizada, respetando su tamaño y ubicación en el diseño del boletín. Estos elementos gráficos fueron configurados para que se mantuvieran consistentes a lo largo de todos los documentos generados.

Adaptación de bibliotecas (por ejemplo, python-docx)

Para lograr la maquetación automática, se utilizó la biblioteca python-docx, que permitió la manipulación de documentos Word de manera sencilla. A través de esta herramienta, fue posible crear tablas con un número dinámico de filas y columnas, dependiendo del contenido de los documentos. Además, se ajustaron parámetros como los bordes de las celdas y el espaciado entre párrafos para garantizar que la presentación final fuera coherente con los estándares institucionales del BOUMH.

Para garantizar una correcta presentación del documento final, también se configuraron estilos específicos para los títulos, subtítulos y cuerpos de texto, siguiendo las normas de formato que se suelen utilizar en los boletines oficiales. Esta personalización de los estilos fue clave para asegurar que el documento tuviera un aspecto profesional y uniforme.

6.5 Pruebas, evaluación y validación

La fase de pruebas fue esencial para garantizar que cada módulo del sistema funcionara correctamente y que el documento final cumpliera con los requisitos de calidad establecidos.

Realización de pruebas de funcionamiento

Se realizaron pruebas exhaustivas de funcionamiento para cada uno de los módulos de traducción y maquetación. Para el módulo de traducción, se verificó la exactitud de las traducciones generadas por Apertium y la API de ChatGPT, evaluando su capacidad para manejar el contexto de las resoluciones rectorales y asegurando que los textos fueran coherentes y fieles al original. Por otro lado, en el módulo de maquetación, se verificó que la distribución simétrica de los textos en ambas lenguas fuera precisa y que los elementos gráficos se insertaran correctamente.

Evaluación comparativa de ambos métodos

Se realizó una evaluación comparativa de los dos métodos de traducción empleados, es decir, la combinación de Apertium con Selenium y la API de ChatGPT. La comparación se hizo en términos de:

- **Precisión**: Evaluación de la calidad lingüística de las traducciones, comprobando la adecuación de los textos en valenciano en cuanto a gramática y contexto.
- **Eficiencia**: Análisis de la rapidez con la que se generaban las traducciones y la capacidad de manejar múltiples documentos en un corto período de tiempo.
- **Viabilidad**: Estudio de la accesibilidad y coste de cada opción, considerando el entorno institucional en el que se aplicará el sistema.

Validación del formato final

Una vez que las traducciones y maquetaciones fueron realizadas, se validó el formato final de los documentos generados en función de los requisitos institucionales del BOUMH. Esto incluyó revisar la correcta distribución de los textos, la consistencia en la presentación de las tablas, y la inclusión de los elementos gráficos en su lugar adecuado. Además, se verificó que el archivo final estuviera correctamente estructurado, con los títulos y subtítulos bien definidos y el texto fácilmente legible tanto en castellano como en valenciano.

6.6 Software y hardware

Descripción del entorno de desarrollo

El sistema fue desarrollado utilizando el entorno de Google Colab, lo que permitió aprovechar los recursos en la nube y ejecutar el código de manera flexible y escalable. El desarrollo se realizó en Python, utilizando diversas bibliotecas especializadas que facilitaron tanto la manipulación de documentos como la traducción automática. Algunas de las bibliotecas clave utilizadas fueron:

- **Selenium**: Para automatizar la interacción con la interfaz web de Apertium y generar traducciones.
- **python-docx**: Para la manipulación de documentos Word, permitiendo crear y modificar tablas y aplicar estilos personalizados.
- **OpenAI API (GPT-4)**: Para la traducción de los textos con un alto grado de contextualización y calidad lingüística.

El uso de Google Colab también permitió realizar pruebas de manera remota y compartir el entorno de trabajo de manera sencilla con otros colaboradores.

Detalles del hardware y recursos computacionales

El sistema se ejecutó en la infraestructura de hardware proporcionada por **Google Colab**, que ofrece acceso a servidores con recursos computacionales en la nube, incluyendo tanto **CPUs** como **GPUs**. Estos recursos fueron clave para llevar a cabo las tareas de traducción automática y la maquetación de los documentos de manera eficiente. Los GPUs de Google Colab fueron especialmente útiles para acelerar los procesos de traducción, ya que los modelos de IA como el de GPT-4 se benefician de la paralelización y la aceleración proporcionadas por las unidades de procesamiento gráfico.

Durante el desarrollo y las pruebas del sistema, se logró manejar un volumen razonable de documentos, y los tiempos de ejecución fueron aceptables. En términos generales, los recursos computacionales disponibles fueron adecuados para el procesamiento de documentos de tamaño medio y grande. Sin embargo, dado que los documentos pueden variar en complejidad y tamaño, se evaluaron

los tiempos de ejecución para garantizar que el sistema pudiera manejar documentos más grandes sin perder eficiencia.

En cuanto a los requisitos de almacenamiento, los archivos generados por el sistema (tanto los documentos originales como las versiones traducidas y maquetadas) se almacenaron de forma temporal en el entorno de Google Colab. Una vez procesados, los documentos finales fueron descargados para su posterior revisión y distribución. Google Colab permitió almacenar los archivos de manera eficiente, sin necesidad de una infraestructura local compleja, y facilitó la descarga de los resultados para su posterior manejo.

Consideraciones

Uno de los aspectos más importantes del sistema fue su **escalabilidad**, es decir, la capacidad de manejar un aumento en el volumen de documentos y adaptarse a futuros requerimientos. El sistema fue diseñado de manera modular y flexible, lo que permitió que fuera fácilmente ampliable para traducir y maquetar un mayor número de documentos. El uso de **Google Colab** y su infraestructura en la nube proporcionó la capacidad de escalar de manera sencilla, aprovechando los recursos computacionales disponibles según las necesidades del proyecto.

A medida que la demanda de traducción y maquetación de documentos aumente, el sistema puede ampliarse para gestionar un mayor volumen de documentos sin necesidad de reestructurar todo el flujo de trabajo. Esto se debe a la capacidad de Google Colab para proporcionar recursos computacionales adicionales cuando sea necesario, y a la naturaleza modular del sistema, que permite integrar nuevas funcionalidades sin afectar su rendimiento general.

Resultados

En cuanto a los resultados obtenidos a partir del desarrollo del sistema automatizado de traducción y maquetación de documentos bilingües para las resoluciones rectorales del Boletín Oficial de la Universidad Miguel Hernández (BOUMH), ambos métodos de traducción implementados han demostrado ser altamente efectivos. Tanto la solución basada en **Selenium con Apertium** como la **API de ChatGPT** han logrado proporcionar traducciones precisas y coherentes, lo que ha permitido cumplir con los objetivos planteados para este

proyecto. A continuación, se detallan los aspectos más destacados de los resultados obtenidos y las diferencias observadas entre ambos enfoques.

Calidad de la Traducción

Ambos métodos han logrado traducir los textos con un alto grado de precisión y adecuación al contexto, lo cual es fundamental dado que los documentos del BOUMH incluyen resoluciones rectorales que deben mantener una redacción formal y clara en ambos idiomas, castellano y valenciano. Las traducciones generadas han sido suficientemente precisas como para ser empleadas en un entorno institucional, cumpliendo con los estándares de calidad lingüística requeridos para estos documentos oficiales.

- **Método Selenium con Apertium**: La solución basada en Selenium automatiza la interacción con la interfaz de Apertium para traducir los textos. Este enfoque ha mostrado buenos resultados, especialmente en términos de rapidez y eficiencia para traducir grandes volúmenes de texto. La traducción proporcionada por Apertium es bastante directa y estructurada, lo que la hace útil en contextos donde se requiere una traducción rápida y precisa. Sin embargo, el estilo de las traducciones tiende a ser un poco más rígido y menos fluido, lo que podría ser una limitación en documentos que requieren una mayor adaptación contextual o estilística.
- **Método API de ChatGPT**: Por otro lado, la API de **ChatGPT** ha demostrado ser más flexible y capaz de generar traducciones más naturales y contextualmente precisas. El modelo GPT-4 tiene una capacidad superior para comprender matices lingüísticos y adaptarse a las particularidades del valenciano, lo que se traduce en traducciones más fluidas, adaptadas a los contextos específicos de los textos. Este enfoque ha proporcionado traducciones de mayor calidad estilística, especialmente en los pasajes que requieren un nivel de sofisticación en el lenguaje o una interpretación más libre de las expresiones.

Diferencias Clave entre los Métodos

A pesar de que ambos métodos han producido resultados satisfactorios, se han observado algunas diferencias significativas en términos de **flexibilidad**, **costo** y **requerimientos de recursos**:

1. **Flexibilidad y Estilo de Traducción**:
    - El **método de traducción con Selenium** es más rígido y estructurado. Al ser un sistema automatizado basado en reglas, la traducción es más cuadriculada y se ajusta estrictamente al contenido original sin demasiada adaptación a las sutilezas lingüísticas del contexto. Esto puede ser beneficioso cuando se necesita una traducción directa y coherente, pero limita la capacidad del sistema para adaptar el texto a un tono más fluido o natural.
    - En contraste, el **método de traducción con la API de ChatGPT** ofrece una mayor flexibilidad y contextualización en las traducciones. Este método es más dinámico, lo que le permite generar traducciones que suenan más naturales, con una mejor adaptación al contexto y al estilo requerido en documentos oficiales. El sistema tiene la capacidad de modificar el tono y la estructura según las necesidades del contenido.
2. **Costo**:
    - Una ventaja significativa del **método con Selenium** es que es completamente gratuito, lo que lo convierte en una opción económica para proyectos con un presupuesto limitado. A pesar de que Apertium ofrece traducciones rápidas y efectivas, el hecho de que no se requiera ningún tipo de inversión adicional lo hace una opción atractiva para las instituciones que necesitan traducir un número elevado de documentos sin incurrir en gastos adicionales.
    - En cambio, la **API de ChatGPT** requiere de una inversión monetaria, ya que es un servicio de pago. Aunque el costo por resolución rectoral es relativamente bajo, el hecho de que se

necesite un presupuesto para acceder al servicio puede ser una limitación para ciertos usuarios. Sin embargo, dado su costo accesible y la calidad superior de las traducciones, muchos usuarios pueden considerar que la inversión vale la pena si se prioriza la calidad y la flexibilidad en la traducción.

Resultados para el BOUMH

Ambos métodos, a pesar de sus diferencias, han cumplido con los requisitos establecidos para el **Boletín Oficial de la Universidad Miguel Hernández (BOUMH)**. Las resoluciones rectorales publicadas en formato bilingüe han sido traducidas con precisión, y los documentos generados han respetado la estructura y el formato necesario para su publicación oficial. El sistema de maquetación, que asegura que los textos se presenten de manera simétrica en ambas lenguas, ha funcionado correctamente en ambos casos, garantizando que los documentos cumplieran con los estándares visuales y estructurales exigidos por la universidad.

- **Método Selenium** ha sido más adecuado para un proceso de traducción simple y rápido, especialmente cuando se requiere manejar una gran cantidad de documentos en poco tiempo. Al ser gratuito, también ofrece la ventaja de reducir costos, lo cual puede ser relevante para un volumen alto de traducciones.
- **Método ChatGPT**, al ser más costoso pero más flexible, ha permitido generar traducciones más adaptadas al contexto institucional, ofreciendo una opción viable cuando se busca una mayor calidad en los textos traducidos. Aunque implica un coste, su rendimiento ha valido la pena cuando la calidad y la fluidez de la traducción son prioritarias.

Conclusión

En conclusión, ambos métodos de traducción han proporcionado resultados satisfactorios para la automatización de las resoluciones rectorales del BOUMH. La elección entre el **método con Selenium** y el **método con ChatGPT** dependerá de las necesidades específicas de la institución, ya sea en términos de presupuesto o de calidad estilística en las traducciones. Ambos enfoques son

válidos y útiles, y el sistema desarrollado ofrece una solución flexible y efectiva para automatizar la traducción y maquetación de documentos bilingües en el contexto institucional del BOUMH.

7. Conclusiones y Propuestas

El desarrollo de un sistema automatizado para la **traducción y maquetación de documentos bilingües** en el contexto de las **resoluciones rectorales** del **Boletín Oficial de la Universidad Miguel Hernández (BOUMH)** ha demostrado ser una solución eficaz para facilitar y agilizar la publicación de documentos oficiales en dos lenguas, castellano y valenciano. A lo largo del proyecto se ha logrado cumplir con los objetivos establecidos, optimizando tanto el proceso de traducción como la presentación visual de los documentos. No obstante, el análisis de los resultados obtenidos ha permitido identificar áreas clave para la mejora y futuras optimizaciones. A continuación, se presentan las principales conclusiones extraídas del proyecto, así como algunas propuestas para el futuro.

Conclusiones

1. **Eficiencia del Sistema Automatizado**: El sistema desarrollado ha demostrado una alta eficacia en la automatización de los procesos de traducción y maquetación. La traducción automática ha sido precisa en la mayoría de los casos, y la integración de los dos métodos de traducción (Selenium con Apertium y API de ChatGPT) ha permitido obtener resultados consistentes y adecuados para las resoluciones rectorales. Ambos enfoques, aunque diferentes en términos de flexibilidad y costos, han permitido cumplir con el objetivo principal de facilitar la publicación de documentos bilingües.

2. **Adecuación a los Requisitos Institucionales**: El sistema ha sido capaz de adaptarse perfectamente a las necesidades del BOUMH, proporcionando una solución que respeta la estructura formal y la distribución simétrica de los textos en castellano y valenciano. La maquetación automatizada ha permitido mantener la coherencia visual de

los documentos, un aspecto crucial en su publicación oficial, garantizando que las traducciones se presenten de manera ordenada y profesional.

3. **Diferencias en los Métodos de Traducción**: El análisis de los métodos de traducción utilizados ha revelado importantes diferencias. Mientras que el **método de Selenium con Apertium** ha sido más adecuado para traducciones rápidas y de bajo costo, el **método de la API de ChatGPT** ha mostrado una mayor flexibilidad y calidad estilística. Esto hace que la elección del método dependa de las prioridades del usuario, ya sea eficiencia económica o calidad de la traducción. En general, la combinación de ambos enfoques podría resultar en una solución más completa, con Apertium utilizado para traducciones masivas y ChatGPT para documentos que requieran un mayor grado de contextualización.

4. **Escalabilidad del Sistema**: El sistema está diseñado para ser escalable, lo que permite manejar un volumen creciente de documentos sin comprometer el rendimiento. La posibilidad de integrar nuevos servicios de traducción o mejorar la interfaz de usuario, junto con la utilización de recursos en la nube, facilita la expansión del sistema según sea necesario. Esta escalabilidad es especialmente relevante dado que la carga de trabajo en el BOUMH puede aumentar con el tiempo, y el sistema debe ser capaz de adaptarse a esa demanda.

5. **Viabilidad del Proyecto en el Contexto Institucional**: El proyecto ha demostrado ser viable desde una perspectiva técnica, ya que se ha logrado desarrollar un sistema funcional que cubre todas las necesidades identificadas en la fase de análisis. Además, se ha confirmado que los documentos traducidos cumplen con los estándares de calidad necesarios para su publicación en el BOUMH. La posibilidad de automatizar todo el proceso de traducción y maquetación representa un ahorro significativo de tiempo y recursos, lo cual es crucial para el funcionamiento eficiente de la institución.

Propuestas de Mejora

Aunque el sistema desarrollado ha cumplido con las expectativas, se han identificado varias áreas en las que se podrían realizar mejoras o actualizaciones

para optimizar aún más el rendimiento y la calidad del proceso de traducción y maquetación:

1. **Optimización del Rendimiento de Traducción**: A pesar de que tanto **Apertium** como **ChatGPT** han proporcionado traducciones adecuadas, existe la posibilidad de mejorar la eficiencia y la precisión de las traducciones mediante el uso de tecnologías adicionales. Una posible mejora sería integrar modelos de traducción automática más avanzados y específicos para el valenciano, lo que podría resultar en una mayor fidelidad a las particularidades del idioma. La implementación de técnicas de **post-edición automática** también podría mejorar la calidad de las traducciones generadas.
2. **Mejora en la Interfaz de Usuario (UI)**: La interfaz de usuario del sistema podría beneficiarse de una actualización para hacerla más intuitiva y accesible para los usuarios no técnicos. Actualmente, la interacción con el sistema puede requerir conocimientos básicos de programación y herramientas de desarrollo. En el futuro, se podría desarrollar una interfaz gráfica de usuario (GUI) que simplifique el proceso de carga de documentos, la selección de métodos de traducción y la visualización de los resultados. Esto facilitaría la adopción del sistema por parte de personal administrativo sin conocimientos técnicos.
3. **Integración de Nuevas Funcionalidades**: Se podrían integrar nuevas funcionalidades al sistema para hacerlo aún más robusto y versátil. Algunas posibles mejoras incluyen:
    - La **creación de una base de datos** que permita almacenar y gestionar documentos traducidos, lo que facilitaría el seguimiento y la recuperación de resoluciones anteriores.
    - La implementación de un **sistema de validación automática** de la calidad de las traducciones, que ayude a detectar posibles errores o incoherencias en las traducciones antes de la publicación de los documentos.

- La **implementación de una herramienta de revisión colaborativa** que permita a los usuarios hacer ajustes en las traducciones generadas por el sistema antes de que los documentos sean finalizados y publicados.

4. **Expansión de la Compatibilidad con Otros Formatos de Documento**: Actualmente, el sistema está orientado a la manipulación de documentos en formato **Word** (con la biblioteca **python-docx**). Sin embargo, sería beneficioso ampliar la compatibilidad a otros formatos de documentos comunes en el ámbito institucional, como **PDF** y **HTML**. Esto permitiría al sistema manejar una mayor variedad de tipos de documentos, ampliando su aplicabilidad en diferentes contextos y situaciones.
5. **Evaluación Continua de la Calidad de la Traducción**: Aunque las traducciones automáticas han mostrado un buen desempeño, sería útil implementar un **sistema de retroalimentación continua** que permita evaluar de manera constante la calidad de las traducciones y realizar ajustes en los modelos de traducción utilizados. Esto podría implicar la recopilación de comentarios de los usuarios para mejorar el sistema y garantizar que las traducciones se mantengan actualizadas y alineadas con los requisitos de la universidad.

Conclusión Final

El sistema desarrollado para la **traducción y maquetación de documentos bilingües** en el contexto de las resoluciones rectorales del **BOUMH** ha sido un éxito en términos de rendimiento y funcionalidad. La automatización de estos procesos ha permitido optimizar tanto el tiempo como los recursos necesarios para la publicación de documentos oficiales, mejorando la eficiencia operativa dentro de la institución.

A pesar de que se han alcanzado los objetivos iniciales del proyecto, existen áreas que podrían beneficiarse de futuras mejoras, tanto a nivel técnico como en términos de usabilidad. Las propuestas de mejora apuntan a aumentar la calidad de la traducción, facilitar la interacción con el sistema y expandir su capacidad para gestionar diferentes tipos de documentos. En resumen, el sistema tiene un gran potencial para evolucionar y seguir aportando valor a la Universidad Miguel

Hernández, convirtiéndose en una herramienta aún más poderosa para la gestión automatizada de documentos bilingües en el ámbito institucional.

Bibliografía

Bibliografía

- Borrás, F. (s.f.)*IA4LEGOS4: Automatización con Python*Universidad Miguel Hernández. https://ia4legos4.umh.es

- Apertium. (s.f.). *Apertium: An open-source rule-based machine translation platform*. https://www.apertium.org

- Google. (s.f.). *Google Colaboratory*. https://colab.research.google.com

- Google. (s.f.). *Google Drive* . https://drive.google.com

- OpenAI. (2024). *OpenAI API documentation* . https://platform.openai.com/docs

- Python Software Foundation. (s.f.). *Python (versión 3.x)* . https://www.python.org

- Selenium. (s.f.). *Selenium WebDriver* . https://www.selenium.dev

- The Apache Software Foundation. (s.f.). *Apache OpenNLP* . https://opennlp.apache.org

- gdown. (s.f.). *gdown: Download large files from Google Drive* . https://github.com/wkentaro/gdown

- python-docx contributors. (s.f.). *python-docx 0.8.11 documentation* . https://python-docx.readthedocs.io

- tqdm developers. (s.f.). *tqdm: A Fast, Extensible Progress Bar for Python* https://tqdm.github.io/

- Python Software Foundation. (s.f.). *Python (versión 3.x)* . https://www.python.org

- Ubuntu. (s.f.). *Ubuntu Keyserver* . http://keyserver.ubuntu.com

- Microsoft. (s.f.). *Office Open XML (docx) File Format Referenc e*. https://learn.microsoft.com/en-us/openspecs/

Anexo

CÓDIGO FUENTE DEL PROYECTO

A continuación se presenta el código fuente completo del sistema de traducción y maquetación de documentos bilingües, el cual utiliza la API de OpenAI para la traducción del castellano al valenciano, y la biblioteca python-docx para manipular documentos Word.

```
# ================== API KEY Y CONFIGURACIÓN INICIAL ==================


# Pega la api_key de OpenAI

clave_api = aquí va la api key'  #@param {type: "string"}


# Enlace a fichero Word en Drive compartido (accesible para cualquiera con el enlace)

link_gdrive = aquí se encuentra el link del texto a traducir'  #@param {type: "string"}


# Extraer el ID del documento de Google Drive

id_gdrive = link_gdrive[35:68]

print("ID del documento:", id_gdrive)


# Descargar el documento Word

!gdown {id_gdrive} -O documento.docx

print("Word file uploaded")
```

```
# ================== INSTALACIÓN DE DEPENDENCIAS ==================

!pip install python-docx openai tqdm

from docx import Document
import openai
import os
from tqdm import tqdm
from time import sleep

# Configuración de la API key de OpenAI en el entorno
%env OPENAI_API_KEY= $clave_api
openai.api_key = os.getenv("OPENAI_API_KEY")

# ================== FUNCIÓN PRINCIPAL DE TRADUCCIÓN ==================

def translate_word_document():
    """
    Traduce un documento Word del castellano al valenciano usando la API de OpenAI.

    El proceso abarca:
      - Cargar el documento original.
```

```
    - Traducir párrafos y tablas manteniendo el formato original.
    - Guardar y descargar el documento traducido.
    """
    # Cargar el documento Word original
    doc = Document("documento.docx")
    new_doc = Document()


    def translate_text(text, max_retries=3):
        """
        Traduce el texto usando la API de OpenAI con manejo de errores y reintentos.
        """
        if not text.strip():
            return text


        for attempt in range(max_retries):
            try:
                response = openai.chat.completions.create(
                    model="gpt-4o-mini",
                    messages=[
                        {
                            "role": "system",
                            "content": (
                                "Ets un Traductor de Valencià, especialitzat en traduir documents del castellà al valencià. "
```

"Els teus principals objectius són garantir traduccions gramaticalment correctes i oferir un text "

"que semble natural i orientat a humans.\n\n"

"Instruccions:\n"

"1. Traduïx el text proporcionat del castellà al valencià.\n"

"2. Assegura't que la traducció mantinga el significat i el context del text original.\n"

"3. Utilitza una gramàtica, sintaxi i expressions idiomàtiques adequades per a fer que la traducció semble natural.\n"

"4. Evita traduccions literals, excepte quan siga necessari per a preservar el significat.\n"

"5. Si hi ha referències culturals o expressions idiomàtiques, adapta-les perquè siguen comprensibles i rellevants en valencià.\n"

"6. Mantín el format i l'estructura del text original, excepte si s'indica el contrari.\n"

"7. Revisa la traducció per a corregir errors o expressions estranyes abans de finalitzar-la.\n\n"

"Característiques addicionals:\n"

"- Capacitat per a traduir entre múltiples idiomes, incloent-hi però no limitant-se a espanyol, francés, alemany, xinés, japonés, àrab, rus i portuguès.\n"

"- Opció per a traduir textos formals i informals de manera adequada segons el context proporcionat.\n"

"- Capacitat per a manejar documents especialitzats, incloent-hi manuals tècnics, textos legals i obres literàries, garantint precisió i rellevància en la terminologia especialitzada.\n\n"

"Exemple:\n"

```
                    "- Text original en castellà: 'La reunión comenzará a las 10 en punto. Por favor, asegúrate de llegar a tiempo.'\n"
                    "- Idioma de destí: Valencià\n"
                    "- Text traduït: 'La reunió començarà a les 10 en punt. Per favor, assegura't d'arribar a temps.'\n\n"
                    "Proporciona el text que vols traduir."
                )
            },
            {
                "role": "user",
                "content": f"Tradueix aquest text al valencià: {text}"
            }
        ],
        temperature=0
    )
    return response.choices[0].message.content

except openai.RateLimitError:
    if attempt < max_retries - 1:
        sleep(20)  # Espera 20 segundos antes de reintentar
        continue
    else:
        raise
except Exception as e:
    if attempt < max_retries - 1:
```

```
                sleep(5)
                continue
            else:
                print(f"Error al traducir texto: {str(e)}")
                return text

print("Iniciando traducción del documento...")

# Procesar cada párrafo del documento
for paragraph in tqdm(doc.paragraphs, desc="Traduciendo párrafos"):
    # Obtener la traducción del párrafo
    translated_text = translate_text(paragraph.text)

    # Crear un nuevo párrafo en el documento traducido
    new_paragraph = new_doc.add_paragraph()
    new_paragraph.style = paragraph.style

    # Insertar el texto traducido y copiar el formato del original
    if translated_text.strip():
        run = new_paragraph.add_run(translated_text)
        for src_run in paragraph.runs:
            run.bold = src_run.bold
            run.italic = src_run.italic
            run.underline = src_run.underline
```

```
    # Procesar cada tabla del documento original
    for table in doc.tables:
        new_table = new_doc.add_table(rows=len(table.rows), cols=len(table.columns))
        new_table.style = table.style

        for i, row in enumerate(table.rows):
            for j, cell in enumerate(row.cells):
                translated_cell = translate_text(cell.text)
                new_table.cell(i, j).text = translated_cell

    # Guardar el documento traducido y descargarlo
    new_doc.save("documento_traducido.docx")
    print("Traducción completada. Documento guardado como: documento_traducido.docx")
    from google.colab import files
    files.download("documento_traducido.docx")


# ================== EJEMPLO DE USO ==================

translate_word_document()
```

Seguidamente tenemos el código fuente del sistema de traducción y maquetación de documentos bilingües con apertium vía selenium.

```
# Añadimos el buscador Debian

cat > /etc/apt/sources.list.d/debian.list <<'EOF'

deb [arch=amd64 signed-by=/usr/share/keyrings/debian-buster.gpg] http://deb.debian.org/debian buster main

deb [arch=amd64 signed-by=/usr/share/keyrings/debian-buster-updates.gpg] http://deb.debian.org/debian buster-updates main

deb [arch=amd64 signed-by=/usr/share/keyrings/debian-security-buster.gpg] http://deb.debian.org/debian-security buster/updates main

EOF


# Añadimos y almacenamos las claves necesarias

apt-key adv --keyserver keyserver.ubuntu.com --recv-keys DCC9EFBF77E11517

apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 648ACFD622F3D138

apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 112695A0E562B32A


apt-key export 77E11517 | gpg --dearmour -o /usr/share/keyrings/debian-buster.gpg

apt-key export 22F3D138 | gpg --dearmour -o /usr/share/keyrings/debian-buster-updates.gpg

apt-key export E562B32A | gpg --dearmour -o /usr/share/keyrings/debian-security-buster.gpg


# Fijamos la APT para conseguir el paquete chromium

cat > /etc/apt/preferences.d/chromium.pref << 'EOF'
```

```
Package: *
Pin: release a=eoan
Pin-Priority: 500


Package: *
Pin: origin "deb.debian.org"
Pin-Priority: 300


Package: chromium*
Pin: origin "deb.debian.org"
Pin-Priority: 700
EOF

# Instalamos chromium y chromium-driver
apt-get update
apt-get install chromium chromium-driver

# Instalamos selenium
pip install selenium # Instalar dependencias necesarias
!pip install python-docx selenium gdown

import requests
```

```
import math as mat
import time
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException
from docx import Document
from docx.oxml.ns import qn
from docx.oxml import OxmlElement
from google.colab import files
import os

# --- Descarga del documento Word desde Google Drive ---
link_gdrive =
'https://docs.google.com/document/d/1YqT8maOADyt268l1rpRSxDrE77CNclWY/edit?usp=sharing&ouid=100433459112157064985&rtpof=true&sd=true'
id_gdrive = link_gdrive.split('/d/')[1].split('/')[0]
print("ID del documento:", id_gdrive)
!gdown {id_gdrive} -O documento.docx
print("Documento Word descargado correctamente.")

# --- Configuración de Selenium con Chromium ---
```

```
chrome_options = Options()
chrome_options.add_argument("--headless")
chrome_options.add_argument("--no-sandbox")
chrome_options.add_argument("--disable-dev-shm-usage")
chrome_options.add_argument("--disable-gpu")
service = Service(executable_path=r'/usr/bin/chromedriver')


def translate_text_apertium(text):
    if not text or text == "0":  # Evitar traducir celdas vacías o con "0"
        return text
    print(f"Intentando traducir: {text[:50]}...")
    start_time = time.time()
    driver = None
    try:
        driver = webdriver.Chrome(options=chrome_options, service=service)
        driver.set_page_load_timeout(20)
        driver.get("https://www.apertium.org/index.spa.html#?dir=spa-
cat_valencia&q=")
        input_textarea = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.XPATH,                '//*[@id="react-
mount"]/div[1]/div[1]/form/div[2]/div[1]/textarea'))
        )
        input_textarea.clear()
        input_textarea.send_keys(text)
        translated_text = WebDriverWait(driver, 10).until(
```

```
            lambda d: d.find_element(By.XPATH, '//*[@id="react-mount"]/div[1]/div[1]/form/div[2]/div[2]/textarea').get_attribute("value").strip() != "",
            message="La traducción no se generó a tiempo."
        )
        translated_text = driver.find_element(By.XPATH, '//*[@id="react-mount"]/div[1]/div[1]/form/div[2]/div[2]/textarea').get_attribute("value")
        print(f"Traducción completada en {time.time() - start_time:.2f} segundos.")
        return translated_text
    except TimeoutException as e:
        print(f"Timeout al traducir el texto: {text[:50]}... Error: {e}")
        return text
    except Exception as e:
        print(f"Error inesperado al traducir el texto: {text[:50]}... Error: {e}")
        return text
    finally:
        if driver:
            try:
                driver.quit()
            except:
                pass
        time.sleep(1)

def set_cell_borders(cell, top=False, bottom=False, left=False, right=False):
    tc = cell._element
```

```
    tcPr = tc.get_or_add_tcPr()
    borders = OxmlElement('w:tcBorders')

    top_border = OxmlElement('w:top')
    top_border.set(qn('w:val'), 'single' if top else 'nil')
    borders.append(top_border)

    bottom_border = OxmlElement('w:bottom')
    bottom_border.set(qn('w:val'), 'single' if bottom else 'nil')
    borders.append(bottom_border)

    left_border = OxmlElement('w:left')
    left_border.set(qn('w:val'), 'single' if left else 'nil')
    borders.append(left_border)

    right_border = OxmlElement('w:right')
    right_border.set(qn('w:val'), 'single' if right else 'nil')
    borders.append(right_border)

    tcPr.append(borders)

def set_table_borders(table):
    """Configura bordes visibles para todas las celdas de una tabla anidada."""
    for row in table.rows:
```

```
        for cell in row.cells:
            tc = cell._element
            tcPr = tc.get_or_add_tcPr()
            borders = OxmlElement('w:tcBorders')

            top_border = OxmlElement('w:top')
            top_border.set(qn('w:val'), 'single')
            borders.append(top_border)

            bottom_border = OxmlElement('w:bottom')
            bottom_border.set(qn('w:val'), 'single')
            borders.append(bottom_border)

            left_border = OxmlElement('w:left')
            left_border.set(qn('w:val'), 'single')
            borders.append(left_border)

            right_border = OxmlElement('w:right')
            right_border.set(qn('w:val'), 'single')
            borders.append(right_border)

            tcPr.append(borders)

def translate_table(original_table, target_doc):
```

```
    print("Traduciendo tabla...")
    rows = len(original_table.rows)
    cols = len(original_table.columns)
    translated_table = target_doc.add_table(rows=rows, cols=cols)

    for i, row in enumerate(original_table.rows):
        for j, cell in enumerate(row.cells):
            original_text = cell.text.strip()
            translated_text = translate_text_apertium(original_text)
            translated_table.rows[i].cells[j].text = translated_text

    set_table_borders(translated_table)  # Añadir bordes visibles
    print("Tabla traducida completada.")
    return translated_table

def copy_table_structure(original_table, target_doc):
    rows = len(original_table.rows)
    cols = len(original_table.columns)
    copied_table = target_doc.add_table(rows=rows, cols=cols)

    for i, row in enumerate(original_table.rows):
        for j, cell in enumerate(row.cells):
            copied_table.rows[i].cells[j].text = cell.text.strip()
```

```
    set_table_borders(copied_table)  # Añadir bordes visibles
    return copied_table

def translate_word_document():
    start_time = time.time()
    doc = Document("documento.docx")
    new_doc = Document()

    paragraphs_list = [p.text.strip() for p in doc.paragraphs if p.text.strip()]
    tables_list = doc.tables
    total_rows = len(paragraphs_list) + 2 * len(tables_list)
    print(f"Total de párrafos: {len(paragraphs_list)}, Total de tablas: {len(tables_list)}, Filas totales: {total_rows}")

    table = new_doc.add_table(rows=total_rows, cols=2)
    row_index = 0

    # Procesar párrafos
    for i, original_text in enumerate(paragraphs_list):
        print(f"Procesando párrafo {i+1}/{len(paragraphs_list)}")
        translated_text = translate_text_apertium(original_text)
        left_cell = table.rows[row_index].cells[0]
        left_cell.text = translated_text
        left_cell.add_paragraph(" ")
        left_cell.add_paragraph(" ")
```

```
        set_cell_borders(left_cell, top=False, bottom=False, left=False, right=True)

        right_cell = table.rows[row_index].cells[1]
        right_cell.text = original_text
        right_cell.add_paragraph(" ")
        right_cell.add_paragraph(" ")
        set_cell_borders(right_cell, top=False, bottom=False, left=False, right=False)
        row_index += 1

    # Procesar tablas
    for i, original_table in enumerate(tables_list):
        print(f"Procesando tabla {i+1}/{len(tables_list)}")
        translated_row = table.rows[row_index]
        translated_row.cells[0].merge(translated_row.cells[1])
        merged_cell_translated = translated_row.cells[0]
        translate_table(original_table, merged_cell_translated)
        set_cell_borders(merged_cell_translated, top=False, bottom=False, left=False, right=False)
        row_index += 1

        original_row = table.rows[row_index]
        original_row.cells[0].merge(original_row.cells[1])
        merged_cell_original = original_row.cells[0]
        copy_table_structure(original_table, merged_cell_original)
```

```
        set_cell_borders(merged_cell_original, top=False, bottom=False, left=False, right=False)
        row_index += 1

    output_filename = "documento_traducido.docx"
    new_doc.save(output_filename)
    print(f"Documento traducido guardado como: {output_filename} en {time.time() - start_time:.2f} segundos")
    files.download(output_filename)

# Ejecutar la función de traducción
translate_word_document()
```