Herramientas de generación de historias narrativas basadas en mapas personalizados
Tools for narrative story generation based on custom maps

Trabajo de Fin de Grado
Curso 2022–2023

Autores
Sam Blázquez Martín
Miguel Hernández García
Alberto Muñoz Fernández

Directores
Gonzalo Méndez Pozo
Pablo Gervás Gomez-Navarro

Grado en Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

Herramientas de generación de historias narrativas basadas en mapas personalizados
Tools for narrative story generation based on custom maps

Trabajo de Fin de Grado en Desarrollo de Videojuegos

Autores
Sam Blázquez Martín
Miguel Hernández García
Alberto Muñoz Fernández

Directores
Gonzalo Méndez Pozo
Pablo Gervás Gomez-Navarro

Convocatoria: *Junio 2023*

Grado en Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

29 de mayo de 2023

Agradecimientos

A nuestras familias y amigos por escucharnos y darnos su apoyo en todo momento hasta el final, tanto en los mejores como en los peores momentos.

Resumen

Herramientas de generación de historias narrativas basadas en mapas personalizados

La generación automática de historias es un campo de estudio dentro del campo de la Inteligencia Artificial que tiene como objetivo generar relatos consistentes y coherentes mediante mecanismos automatizados. En este trabajo se pretende generar una herramienta capaz de crear historias narrativas con suficiente profundidad para ser útiles para escritores y creadores de todos los ámbitos, a partir de un mapa geográfico/político. Un escritor creador de mundos normalmente cuenta con una idea general de lo que quiere conseguir, así como un mapa de su mundo, pero puede ser tedioso pensar en distintas historias para cada uno de sus territorios. Nuestro proyecto pretende ofrecer una forma de generar una primera lluvia de ideas de forma totalmente automática y veloz.

En este proyecto realizamos un estudio de los distintos avances en el campo del aprendizaje automático, modelos de lenguaje y distintas técnicas relacionadas, generadores de mapas presentes en el mercado, así como generadores de historias. Mostraremos como estos últimos no presentan la posibilidad de utilizar un mapa para generar historias, opción que nuestro proyecto busca ofrecer.

Para la realización de este programa concebimos un plan analizando las distintas tecnologías del mercado. Posteriormente, realizamos un primer procesado de los datos del mapa y un primer prototipo. Al no conseguir el resultado esperado realizamos una mejora de dicho prototipo mediante distintos mecanismos, que incluyen: una puntuación de los resultados obtenidos para seleccionar el texto con mejor nota, prevención de elementos indeseados para un texto narrativo histórico, *fine-tuning* de uno de los modelos e investigación sobre mecanismos de cohesión.

Los resultados obtenidos al final del proyecto son mucho mejores de los obtenidos en nuestro primer prototipo, lo que demuestra la eficacia de nuestros mecanismos. Sin embargo, el trabajo realizado es altamente ampliable, pudiendo obtener muchos mejores resultados con la utilización de modelos mucho más modernos y actuales a los utilizados en nuestro desarrollo. Los mecanismos desarrollados hasta ahora mejorarán aún más cualquier resultado obtenido, permitiendo continuar en el futuro con la parte del proyecto correspondiente a la cohesión entre historias.

Palabras clave

Generación de mapas, generación de historias, IA, LLMs, fine-tuning

Abstract

Tools for narrative story generation based on custom maps

Automatic story generation is a field of study within the field of Artificial Intelligence that aims to generate consistent and coherent narratives through automated mechanisms. This project intends to generate a tool capable of generating narrative stories with enough depth to be useful for writers and creators from various domains, based on a geographic/political map. A world-building writer typically has a general idea of what they want to achieve, as well as a map of their world, but it can be tedious to think of different stories for each territory. Our project aims to provide a way to generate an initial brainstorming session automatically and swiftly.

In this project, we conducted a study of the different advancements in automatic learning, language models and related techniques, market-available map generators, as well as story generators. We will demonstrate how the latter do not offer the possibility of using a map to generate stories, which is an option that our project seeks to provide.

To develop this program, we devised a plan by analyzing the different technologies available in the market. Subsequently, we performed an initial processing of the map data and created a first prototype. As we did not achieve the expected results, we improved the prototype using various mechanisms, including: scoring the obtained results selecting the result with the best score, prevention of undesired elements for a historical narrative text, fine-tuning one of the models, and researching cohesion mechanisms.

The results obtained at the end of the project are much better than those obtained in our first prototype, which demonstrates the effectiveness of our mechanisms. However, the work carried out is highly expandable, as much better results can be obtained by using much more modern and up-to-date models than those used in our development. The mechanisms developed so far will further enhance any achieved result, allowing for the continuation of the project's cohesion between stories in the future.

Keywords

Map generation, story generation, AI, LLMs, fine-tuning

Índice

Índice de figuras

Capítulo 1

Introducción

Este último año, las IAs han sido muy a menudo el tema principal de conversación entre personas relacionadas con la informática, ya que están en constante crecimiento y han sido creados infinidad de modelos muy avanzados. Desde modelos como Dall-e, un sistema de IA que puede crear imagenes realistas y arte con tan solo una descripción escrita en lenguaje natural (OpenAI (2021)), o los modelos de Midjourney, un equipo de investigación independiente centrados en el diseño, la infraestructura humana e Inteligencia Artificial, que ha desarrollado varios modelos diferentes que giran entorno a la generación de imágenes (Midjourney (2022)). Por otro lado, también existen IAs como ChatGPT, un modelo entrenado que interactua de forma conversacional y es capaz de responderte una infinidad de preguntas, admitir sus errores, y desafiar premisas incorrectas (OpenAI (2022)).

Existen diversos modelos de generación de texto narrativo centrado en la creación de novelas, cuentos y similares.Sin embargo, hemos observado la falta de un modelo exclusivamente enfocado en la generación de mundos narrativos. Es por ello que nos preguntamos si sería posible hacer uso de uno de los existentes para crear una historia que incluya información de civilizaciones, religiones, culturas, conflictos y demás partiendo de un mapa. En consecuencia, nos sentimos atraidos por la idea de crear un sistema capaz de crear historias con profundidad sobre civilizaciones, partiendo de un mapa detallado donde encontramos información socio-pólitica a través del uso de modelos ya existentes de generación narrativa. Surgiendo así la idea base de nuestro proyecto, el desarrollo de una herramienta que tome ciertos parámetros personalizables por el usuario, y cree una historia dirigida por éste lo suficientemente completa como para hacer uso de ella, o sirva de inspiración creativa para el desarrollo de otros proyectos.

1.1. Motivación

Este proyecto pretende ofrecer a todo aquel que lo necesite la creación de una historia narrativa. Existen diversas situaciones en las que la creación de un mundo narrativo puede resultar útil y significativa, como por ejemplo, un escritor novel podría necesitar un mundo en el que escribir una historia, o un artista buscar inspiración para un dibujo, o un master de rol necesitar de un mundo en el que situar

su partida; y nuestra herramienta pretende ser útil para todas estas situaciones, inspirar y dar una base creativa a todo aquel que lo necesite.

Las historias creadas no tienen intención de ser obras, ni sustituir un trabajo humano, sino generar información sobre una serie de civilizaciones, culturas y religiones para que formen la base de un proyecto todavía mayor. Ya que no se detalla en profundidad ningún evento ocurrido, ni se habla de personajes en concreto, se ofrece unicamente una estructura básica socio-política a la cual poder darle un uso más concreto deseado por el usuario.

1.2. Objetivos

El objetivo del proyecto consiste en desarrollar una herramienta que permita la generación de historias narrativas coherentes y entrelazadas entre sí a partir de un mapa geográfico/político. Esta herramienta incluirá la representación de diversas civilizaciones, culturas, religiones, y otros elementos que contribuyan a la riqueza y coherencia de las historias generadas. El mapa de origen debe de poder ser configurable a gusto del usuario de forma previa a la utilización de nuestro programa, permitiendo el mayor grado posible de libertad creativa. El programa resultante respetará dichos parámetros introducidos por el usuario, generando una serie de historias consistentes con las generadas previamente por la misma herramienta. El resultado de las generaciones será mostrado al usuario en forma de texto plano.

La generación de historias utilizará modelos de lenguaje gratuitos que permitan el uso del programa por cualquier tipo de usuario. Además, se implementarán distintas herramientas de corrección de sintaxis y herramientas de consistencia narrativa para ofrecer mejores resultados con cualquier modelo utilizado.

1.3. Metodología

En el desarrollo de nuestro proyecto, hemos aplicado metodologías ágiles, concretamente Scrum, para asegurar un enfoque eficiente y flexible en la planificación y ejecución de las tareas. Inicialmente durante la fase de investigación, considerando nuestras limitaciones debido a compromisos académicos, establecimos metas de investigación por períodos de dos a tres semanas, donde nos enfocamos en la exploración y comprensión de los conceptos más relevantes para el proyecto.

A partir de enero, cuando nos centramos en el desarrollo de la herramienta, ajustamos los tempos y adaptamos nuestra metodología a *sprints* semanales, conocidos como *weeklys*, con el objetivo de avanzar de manera más rápida y efectiva. Durante estos *sprints*, asignamos tareas específicas a cada miembro del equipo y nos reunimos regularmente para revisar el código y los avances individuales, lo cual nos permitió mantenernos al tanto del proyecto en su totalidad y comprender su funcionamiento integral.

Además, hemos realizado una estimación inicial del trabajo que refleja la distribución de las tareas a lo largo del tiempo, pero nos adaptamos de manera flexible a medida que avanzamos, siguiendo los principios de Scrum. Durante el desarrollo

del proyecto, evaluamos constantemente las necesidades y prioridades cambiantes, ajustando la distribución de las tareas a lo largo del tiempo. Teniendo en cuenta nuestros compromisos académicos y otras limitaciones, hemos establecido objetivos realistas y alcanzables en cada etapa del proyecto. Esto nos ha permitido mantener un ritmo constante y garantizar un progreso constante hacia nuestros objetivos.

Para la creación y organización de las tareas del proyecto, así como la asignación de las mismas, decidimos hacer uso de Discord. Discord es un programa de comunicación y socialización remota y gratuita que incorpora chats de texto, de voz y vídeo. En dicho programa los usuarios pueden crear servidores e invitar a otros usuarios a participar en los canales de voz o de texto de los que este disponga (Schwartz (2021)). Hemos creado un servidor para el desarrollo del trabajo, pudiendo así participar en los diferentes canales que hemos creado para seccionar la información, disponiendo de un canal de bibliografía, avances realizados, erratas de la memoria...De esa manera estructurábamos bien las metas e investigaciones y evitábamos fugas donde perdíamos enlaces a investigaciones o trabajo.

Hemos realizado una estimación de la planificación del trabajo en la Tabla 1.1 teniendo en cuenta nuestras limitaciones con el avance del proyecto debido a asuntos académicos.

En resumen, hemos aplicado metodologías ágiles, en particular Scrum, en nuestro enfoque de desarrollo. Hemos realizado una planificación cuidadosa y hemos adaptado nuestra metodología a medida que avanzábamos en el proyecto, permitiéndonos optimizar nuestro tiempo y recursos, así como mantenernos alineados y comprometidos con nuestros objetivos.

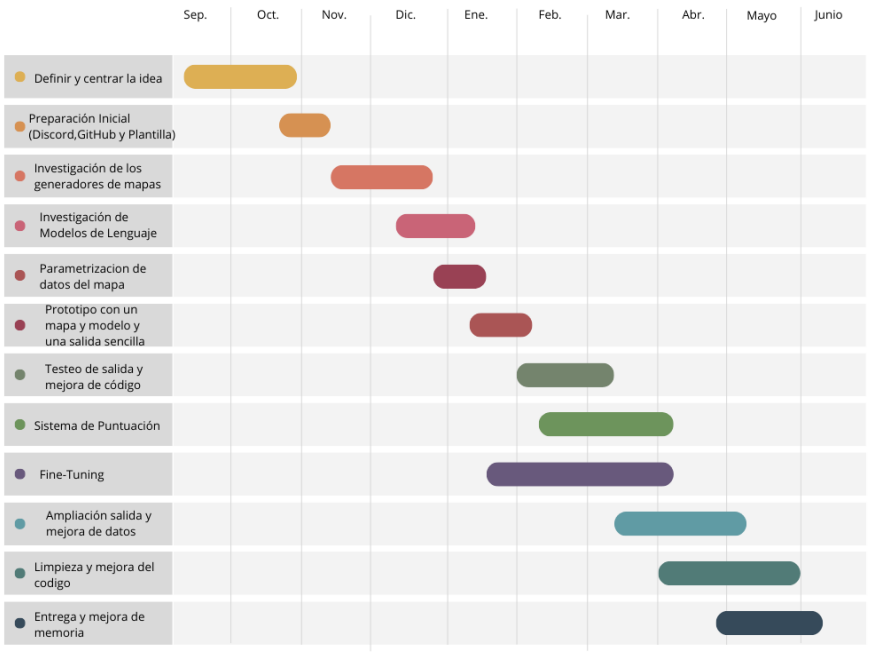

Figura 1.1: Planificación del trabajo a realizar desde el comienzo del curso hasta la fecha de entrega

1.4. Plan de trabajo

Para cumplir los objetivos mencionados anteriormente, dividimos el trabajo de la siguiente manera para comenzar el desarrollo:

1. Buscar una forma sencilla de generar un mapa, con características personalizables y al mismo tiempo encontrar una manera de parametrizarlas, ya sea con un generador o una herramienta que sea capaz de leerlos, y nos permita exportar los datos en un formato fácilmente legible por una persona, y útil para el proyecto.

2. Hacer un estudio de los diferentes Modelos de Generación de Lenguaje Natural presentes actualmente de manera pública, que puedan ser utilizados para la generación de historias partiendo de una entrada de datos, y seleccionar los que más beneficios nos puedan aportar a nuestro proyecto.

3. Realizar la lectura del mapa generado, almacenar los datos en una estructura, seleccionar los más relevantes para la creación de la historia, y transformarlos en una entrada procesable por el modelo.

4. Cargar los modelos previamente seleccionados para el proyecto, crear prototipos para una entrada de datos inicial que sea simple y realizar una comparativa de las salidas que nos proporcionen cada uno, y elegir el modelo con mejores resultados.

5. Aumentar la calidad de las salidas elaboradas por los modelos, siendo así más desarrolladas y completas, a través de la aportación de una mayor variedad de entradas para evitar la repetición, además de, aumentar la cantidad de características del mapa que se emplean y modificar los parámetros del modelo para aumentar la longitud del texto generado.

6. Crear un sistema de evaluación que dadas un conjunto de salidas, escoja la mejor formulada, para evitar posibles errores y aumentar la consistencia del texto.

7. Iterar sobre los dos pasos anteriores para refinar el proyecto y mejorar periodicamente los resultados obtenidos, y seleccionar un grupo de personas que nos aporten retroalimentación.

8. Generar un documento de texto con las historias creadas por nuestra herramienta.

1.5. Estructura del documento

Esta memoria está estructurada en diversos apartados, que se detallarán a continuación:

1. Como ya se ha visto, en el capítulo 1 hacemos un resumen general del proyecto mencionando el propósito, nuestras metas con la creación de esta herramienta y el contexto en el que se puede utilizar.

2. En el capítulo 2 se indica toda la investigación necesaria sobre los generadores de mapas, los distintos modelos de lenguaje para el procesamiento del texto, la investigación de una técnica de machine learning para adaptar uno de los modelos de lenguaje seleccionados llamada Fine-Tuning y una investigación acerca de otros generadores de historias alternativos ya presentes en el mercado.

3. En el capítulo 3 hablamos sobre el desarrollo de la herramienta y cómo tuvimos que seccionar el trabajo para avanzar de manera simultánea. Por una parte probando y seleccionando los modelos de lenguaje mencionados en el capítulo 2 y comparando sus salidas para escoger el mejor resultado. Por otra viendo las limitaciones de los generadores de mapas y la manera de parametrizar toda su información y guardarlo en un grafo en el proyecto. Y por otra la implementación de un sistema de puntuación que evalue la salida de los modelos para generar el mejor resultado. También se mencionan unas mejoras de las salidas del modelo para aumentar la cantidad de texto que generaba, además de la implementación del Fine-Tuning mencionado en el capítulo 2 para uno de los modelos seleccionados.

4. En el capítulo 4 hablamos de los resultados obtenidos con el último prototipo de nuestra herramienta y cómo éstos han influenciado al desarrollo de la versión final del proyecto.

5. En el capítulo 5 se discute sobre los resultados obtenidos gracias a nuestra herramienta en comparación con otro tipo de soluciones, así como las limitaciones y las ventajas encontradas que puede tener.

6. Por último, en el capítulo 6, damos una conclusión a este trabajo, hablando tanto de las metas cumplidas como de las limitaciones que hemos tenido, mencionando los problemas que nos hemos encontrados y como se podría mejorar esto de cara al futuro, además de unas posibles ampliaciones del proyecto.

Capítulo 2

Estado de la Cuestión

En este capítulo vamos a hablar de todo el proceso de investigación previo antes de empezar a realizar el trabajo. Dicha investigación incluye un análisis extenso acerca de distintos generadores de mapas disponibles para el público general, una investigación sobre diversos generadores de historias, una enumeración de distintos modelos comúnmente utilizados en dicha generación de historias, así como una investigación previa acerca de modelos de lenguaje y el proceso de fine-tuning para mejorar la salida del modelo.

2.1. Generadores de mapas presentes en el mercado

En primer lugar, nuestro proyecto precisaba de una fuente fiable de donde poder obtener mapas fácilmente personalizables por el usuario. Una herramienta generadora de mapas se trata de un software capaz de ayudar al usuario a generar mapas considerando distintas características. Existen mapas de distintos tipos, por ejemplo: políticos, geográficos, de calor, del tiempo...

En nuestro caso lo que buscamos es un mapa que sea capaz de representar tanto civilizaciones, como sus distintas culturas, religiones y relaciones, así como una diferenciación entre los distintos climas y accidentes geográficos presentes.

Para ello realizamos una investigación exhaustiva de las distintas herramientas gratuitas que cualquier usuario podría utilizar libremente con el objetivo de introducir posteriormente dicho mapa en nuestro programa. Además, también tuvimos en cuenta la popularidad de dichas herramientas, así como su soporte a largo plazo.

2.1.1. Donjon

Donjon[1] se trata de una página web que cuenta con una colección de herramientas y generadores cuya utilización se encuentra centrada en sistemas de rol.

Un sistema de rol es una serie de reglas empleadas en la realización de partidas de rol, un tipo de entretenimiento donde un grupo de jugadores se reúnen para interpretar el rol o la persona de un personaje ficticio en un mundo y una historia

[1] `https://donjon.bin.sh`

usualmente ficticias (Mackay (2017)). En dicha partida suele haber un Game Master o GM, que se encarga del desarrollo del mundo de la partida, los personajes que no son controlados por los otros jugadores, además de la organización de dicha partida. La presencia de un sistema definido permite que la realización de dicha partida sea justa y disfrutable para todos los jugadores y el GM.

Donjon no solo permite generar mapas geográficos, sino que también permite generar personajes, nombres, aventuras, etc. De una manera altamente sencilla.

2.1.2. Inkarnate

Inkarnate[2] es uno de los generadores de mapas más famosos que se adaptan a las características buscadas. Cuenta con una gran capacidad de personalización, donde el usuario puede crear continentes, mares, lagos, formaciones montañosas, poblados, con total facilidad. Todo esto mediante una cantidad inmensa de recursos, para conseguir un mapa de forma rápida y sencilla.

Además, cuenta con una comunidad muy activa y actualizaciones frecuentes. Gran parte de sus características se encuentran limitadas y precisan de que el usuario cuente con una suscripción activa para desbloquear.

2.1.3. Wondercraft

Wondercraft[3] es también uno de los generadores de mapas más conocidos, sin embargo, a diferencia de Inkarnate su servicio es de pago único, ofreciendo una gran alternativa para aquellos usuarios que prefieran realizar un pago único.

Presenta una gran diferencia en el estilo de los mapas resultantes respecto a Inkarnate, dando una impresión más cartográfica y antigua, pero sus características, opciones ofrecidas y posibilidades son muy similares. Además, aporta herramientas específicas de relieve del terreno, las cuales no son soportadas por Inkarnate.

2.1.4. Azgaar's Fantasy Map Generator

Un generador altamente personalizable, cuenta con una generación inicial aleatoria, pero permite que el usuario edite a su gusto cualquier parte del mapa posteriormente[4]. Además, incluye una gran variedad de características como relaciones civilizaciones, culturas, ciudades, etc. Todo ello con un nivel de exhaustivo de detalle y totalmente gratuito. Este generador es de código abierto, y cuenta con una comunidad de más de 2500 personas en su servidor de Discord.

Posee además una interfaz muy cómoda e intuitiva, además de permitir guardar tu mapa en un fichero para poder seguir editandolo posteriormente. Todas las nuevas versiones del generador son compatibles con los mapas generados en versiones anteriores, lo que permite que nunca pierdas tu trabajo.

[2] `https://inkarnate.com`
[3] `https://www.wonderdraft.net`
[4] `https://azgaar.github.io/Fantasy-Map-Generator/`

2.2. Modelos de Lenguaje

Para comenzar a hablar sobre generación de historias, es necesario entender algunos conceptos fundamentales, como el campo de las Inteligencias Artificiales, apredizaje automático, deep learning y finalmente,los modelos de lenguaje. La IA es un campo complejo que engloba la habilidad de una máquina de presentar las mismas capacidades que los seres humanos, como el razonamiento, el aprendizaje, la creatividad y la capacidad de planear (Parlamento Europeo (2021)). En este contexto, consideraremos a la IA como una disciplina y conjunto de capacidades que busca desarrollar máquinas capaces de imitar la inteligencia humana. La investigación en IA ha dado lugar a diferentes tipos de Inteligencias Artificiales o máquinas entrenadas para imitar la inteligencia humana, las cuales utilizan diversos modelos para lograr este objetivo.

El aprendizaje automático[5] es una rama de la inteligencia artificial que se centra en el desarrollo de algoritmos y técnicas que permiten a las máquinas aprender de los datos sin ser programadas explícitamente. En lugar de seguir instrucciones específicas, los modelos de aprendizaje automático son capaces de reconocer patrones y tomar decisiones basadas en la información proporcionada.Dentro del aprendizaje automático, se encuentra el *deep learning* (aprendizaje profundo)(Rouhiainen (2018)), que es una subcategoría que se ha vuelto muy influyente en los últimos años. El *deep learning* se basa en redes neuronales artificiales con múltiples capas que imitan el funcionamiento del cerebro humano. Estas redes son capaces de aprender y extraer características relevantes de los datos de entrada, lo que les permite realizar tareas más complejas, como el reconocimiento de imágenes, el procesamiento del lenguaje natural y la toma de decisiones.

Por último, los modelos del lenguaje son redes neuronales artificiales de gran tamaño, capaces de analizar ingentes volúmenes de texto escrito para aprender la estructura con la que se presentan las palabras de un determinado idioma (Serrano et al. (2022)). Su objetivo principal es estudiar la relación entre los sistemas informáticos y el lenguaje natural, buscando establecer un mecanismo de comunicación entre personas y máquinas. Es una forma de Inteligencia Artificial (Wikipedia (2023b)) entrenada para generar o comprender lenguaje natural.

Los modelos de lenguaje están estrechamente relacionados con el aprendizaje automático y el *deep learning*. En el ámbito del procesamiento del lenguaje natural, éstos son una aplicación directa del aprendizaje automático.

Transformers

Durante nuestra investigación, descubrimos un tipo de modelo llamado Transformer (Wikipedia contributors (2023c)). Los Transformers son un tipo de redes neuronales que introducen una serie de bloques atencionales[6]. Estos bloques capturan relaciones entre palabras en una oración en un texto. Esto permite al modelo considerar el contexto de cada palabra dentro de la oración completa, en lugar de

[5] https://www.hpe.com/lamerica/es/what-is/machine-learning.html
[6] https://twitter.com/jorgemcalvo/status/1661744043838279680?s=20

hacerlo de forma secuencial. Esta implementación permite capturar relaciones a largo plazo y comprender las dependencias contextuales en el texto, procesando las palabras simultáneamente.

Este concepto de atención mencionado previamente se trata de una medida de relevancia de una palabra en la oración. A cada palabra se le asocia un valor de atención distinto dependiendo de esta importancia. Estas puntuaciones son utilizadas posteriormente para ponderar las palabras y formar una contextualización de la oración. Por ello, las palabras son capaces de obtener información de las palabras colindantes, así como de su contexto en la oración. Diferentes modelos cuentan con diferentes tamaños para este bloque atencional, lo que limita o amplia la cantidad de palabras con las que la palabra objetivo es capaz de contextualizarse.

Los modelos de lenguaje hacen uso de un proceso denominado tokenización para utilizar estos bloques atencionales. Tokenizar consiste en el proceso de separar las palabras en tokens, es decir, identificarlas como palabras que ya tienen registradas. Diferentes modelos cuentan con un número de tokens registrados diferente, así como una capacidad de análisis de bloque atencional distinto.

Todos estos factores vuelven a este tipo de modelos un recurso indispensable en la generación de lenguaje natural, un campo donde el contexto y el análisis simultaneo de información es totalmente esencial.

Al no haber trabajado nunca con ningún transformer tuvimos que investigar acerca de su funcionamiento interno, así como su modelo de arquitectura y elementos clave para utilizarlo correctamente (Markowitz (2023)). Es importante tener en cuenta que los modelos de lenguaje generativos basados en autoregresión como los Transformers tienen una mayor tendencia de sufrir *Word Looping* y *Looping Behavior* (Holtzman et al. (2019)). El *Word Looping* o *Looping Behavior* es un fenómeno presente en el texto generado por algunos modelos de lenguaje que contiene una repetición excesiva de palabras o frases. Este fenómeno puede ocurrir cuando el modelo entra en un bucle donde repite sistemáticamente la misma secuencia de palabras, lo que resulta en una falta de diversidad y originalidad en las salidas generadas. Estos fenómenos pueden ser problemáticos, ya que afectan la calidad y coherencia del texto producido. Más adelante hablaremos sobre cómo creamos el sistema de puntuación para evitar así esos problemas. Los modelos descritos a continuación cumplen con esos requisitos y se encuentran ordenados cronológicamente en el orden en el que fueron lanzados al mercado.

2.2.0.1. BERT

BERT (Bidirectional Encoder Representations from Transformers), fue uno de los modelos desarrollados por Google AI[7] con un rendimiento de vanguardia en el año 2018, siendo una gran revolución en el campo de la Inteligencia Artificial en tareas de comprensión del lenguaje natural (Wikipedia (2023a)). Fue muy potente y prominente en su día, batiendo una gran cantidad de récords en pruebas de la época (Devlin et al. (2018)).

La principal innovación de BERT radica en su capacidad para capturar la bidireccionalidad en el procesamiento del lenguaje. A diferencia de los modelos anteriores,

[7] `https://ai.google`

que se basaban en un enfoque unidireccional, BERT permite que el contexto de las palabras se propague en ambas direcciones. Esto significa que BERT puede capturar las dependencias contextuales tanto de las palabras anteriores como de las posteriores en una oración, lo que mejora significativamente la comprensión del texto.

Para lograr esto, BERT utiliza una fase de pre-entrenamiento y una fase de ajuste. Durante la fase de pre-entrenamiento, el modelo se entrena en grandes cantidades de datos textuales no etiquetados, como textos de Wikipedia o libros, para aprender representaciones generales del lenguaje. Durante este proceso, el modelo se expone a tareas de lenguaje enmascaradas, donde se ocultan palabras en una oración y el modelo debe predecirlas en función del contexto circundante.

Una vez que el modelo ha sido pre-entrenado, se lleva a cabo la fase de ajuste donde se adapta el modelo a tareas específicas de NLP. Esto implica entrenar el modelo en conjuntos de datos etiquetados para realizar tareas como clasificación de texto, reconocimiento de entidades o respuesta a preguntas. Durante el ajuste, el modelo ajusta sus pesos y parámetros para adaptarse a la tarea específica, utilizando la información aprendida durante la fase de pre-entrenamiento.

En resumen, BERT es un modelo que captura la bidireccionalidad en el procesamiento del lenguaje. A través de su fase de pre-entrenamiento y posterior ajuste, BERT puede aprender representaciones del lenguaje altamente efectivas y aplicar ese conocimiento a tareas específicas de NLP.

2.2.0.2. GPT-2

Introducido al mercado y desarrollado por OpenAI en 2019, GPT-2 (Generative Pre-trained Transformer 2) destacaba en su gran cantidad de aplicaciones. Una de las características sobresalientes de GPT-2 es su capacidad para generar texto continuo y de calidad que se asemeja al lenguaje humano. El modelo puede completar oraciones, escribir artículos, generar historias o incluso mantener conversaciones. Además, GPT-2 permite el control del estilo y el tono del texto generado, lo que lo hace versátil y adaptable a diversas aplicaciones.

A diferencia del resto de modelos, GPT-2 tiene capacidad de generación autónoma de texto, haciéndolo muy útil en aplicaciones como la generación automática de historias, la creación de contenido de marketing u otras tareas donde se requiere una producción de texto creativa y fluida.

Sin embargo, es importante tener en cuenta que GPT-2 tiene limitaciones y desafíos. Pese a que haya sido entrenado con grandes cantidades de datos de texto, ésto mismo hace que se pueda generar información falsa o engañosa si no se controla adecuadamente. También puede mostrar cierta falta de coherencia y comprensión contextual en situaciones complejas (Wikipedia contributors (2023b)).

Actualmente se puede acceder a GPT-2 de manera gratuita, por lo que nos podría ser muy útil al no necesitar ningún recurso o sistema de pago.

2.2.0.3. GPT-3

Uno de los transformers más precisos hasta la fecha. Al igual que su antecesor (GPT-2), cuenta con la capacidad de poder ser utilizado para una gran cantidad

de aplicaciones. Este modelo fue un gran avance en el año 2020 en la generación computacional de lenguaje natural. También siendo capaz de codificar en una gran variedad de lenguajes de programación, dio lugar a una gran variedad de aplicaciones más allá de la generación de texto narrativo o conversacional. El texto era tan similar al producido por una persona que los propios desarrolladores avisaron de los posibles riesgos que el uso de semejante tecnología podría provocar (Dale (2021)). GPT-3 se destaca por su entrenamiento con un conjunto de datos enormemente ampliado, contando con aproximadamente 175 billones de parámetros, en comparación con los 1.5 mil millones de GPT-2. Este aumento masivo de parámetros permite que GPT-3 capture relaciones complejas, se adapte a diversos dominios y estilos, mejore la coherencia y precisión en la generación de texto, realice razonamiento contextual y fomente la creatividad en la creación de contenido. Es un cambio significativo que impulsa su capacidad para comprender y generar texto de alta calidad en comparación con su predecesor.

2.2.0.4. MarIA

MarIA fue un modelo desarrollado por la Biblioteca Nacional y el BSC (Barcelona Supercomputing Center) (Gutiérrez-Fandiño et al. (2022)). Está especializado en la generación de texto en la lengua española. Basado en la arquitectura de GPT-2, MarIA tiene la capacidad de generar texto a partir de un ejemplo previo, lo que la hace útil en la elaboración de resúmenes y en la simplificación de información[8].

El entrenamiento del modelo se realizó utilizando archivos de la propia Biblioteca Nacional, los cuales fueron rigurosamente revisados y corregidos para evitar posibles errores de información en el modelo. Se requiere investigación adicional para evaluar y explorar en profundidad las capacidades y posibilidades de MarIA en diferentes aplicaciones y contextos.

2.2.0.5. BETO

BETO es un modelo de lenguaje basado en BERT (Bidirectional Encoder Representations from Transformers) diseñado específicamente para el procesamiento del lenguaje natural en español. Este modelo ha sido entrenado con un gran corpus de texto en español y se ha convertido en una herramienta poderosa para tareas de procesamiento del lenguaje en este idioma[9].

Una de las características destacadas de BETO es su capacidad para comprender y generar texto en español con las características propias de este idioma, como el uso de tildes y la "ñ". Esto lo convierte en una herramienta altamente relevante y valiosa para aplicaciones y proyectos que requieren un procesamiento del lenguaje de alta calidad en español.

BETO ha sido desarrollado por un equipo de investigadores y expertos en procesamiento del lenguaje natural, y su disponibilidad ha contribuido significativamente al avance y desarrollo de aplicaciones y soluciones en español. Sin embargo, es importante tener en cuenta que, a pesar de que se ha entrenado con una cantidad con-

[8] https://portal.mineco.gob.es/es-es/comunicacion/Paginas/211111_np_maria.aspx
[9] https://medium.com/dair-ai/beto-spanish-bert-420e4860d2c6

siderable de datos, los modelos de lenguaje en español se enfrentan a la limitación de no contar con un volumen suficiente de datos de calidad para su entrenamiento. Esta es una área en la que se requiere una mayor investigación y recopilación de datos para seguir mejorando y optimizando los modelos de lenguaje en español.

2.2.0.6. GPT-Neo

GPT-Neo es un modelo de lenguaje de código abierto que se presenta como una alternativa a GPT-2, ofreciendo una opción más accesible para su utilización en investigaciones y otros ámbitos. Este modelo ha sido entrenado utilizando los datos disponibles en la base de datos de The Pile[10], que es un conjunto de datos de código abierto y diverso con un tamaño aproximado de 825 GiB.

El hecho de que sea de cógido abierto conlleva varias ventajas significativas. En primer lugar, permite contar con una comunidad activa de usuarios y desarrolladores que pueden contribuir con nuevas funcionalidades, mejoras y corrección de errores, lo que impulsa su desarrollo continuo y su adaptación a diferentes necesidades. Además, al tener acceso al código fuente, los usuarios tienen la posibilidad de entrenar el modelo con datos personalizados, lo que brinda una mayor flexibilidad y adaptabilidad para abordar tareas específicas.

2.3. Fine-tuning

Como parte de nuestra investigación, nos informamos acerca de procesos comúnmente realizados en la utilización de modelos. Existe un proceso denominado *fine-tuning* (Wikipedia contributors (2023a)) que consiste en la modificación de los pesos de un modelo mediante el entrenamiento del modelo pre-entrenado con nuevos datos. Con el fin de proporcionar un mayor contexto, explicaremos a continuación de forma simplificada el funcionamiento de un modelo en el campo de la Inteligencia Artificial y el concepto de los pesos en referencia a un modelo de Inteligencia Artificial.

Un modelo es una red neuronal, es decir una red conectada de neuronas. Estas solo pueden tener asignado un valor numérico de 0 o 1 (representando el funcionamiento de una neurona biológica que pueden encontrarse excitadas o no). Como se puede observar el la figura 2.1[11], se encuentran organizadas en capas. La primera capa de la red se denomina siempre capa de entrada y la última capa es denominada como capa de salida. También es posible que estén presentes otras capas entre estas dos, denominadas capas ocultas. Todo nodo de una capa se encuentra conectado con todos los nodos de la capa posterior, excepto los nodos presentes en la capa de salida. Dicha conexión determina si la siguiente neurona se encontrará excitada o no, esta posibilidad es definida por una función que produce un valor entre 0 o 1 usualmente, determinando 0 un 0 por ciento de probabilidad, y 1 un 100 por ciento de posibilidad. Cuando un dato entra en esta red, el dato será evaluado por todas

[10] `https://huggingface.co/docs/transformers/model_doc/gpt_neo`

[11] Página web donde se encuentra la figura 2.1 `https://commons.wikimedia.org/w/index.php?curid=5084582`

las neuronas desde la capa de entrada hasta llegar a la capa de salida.

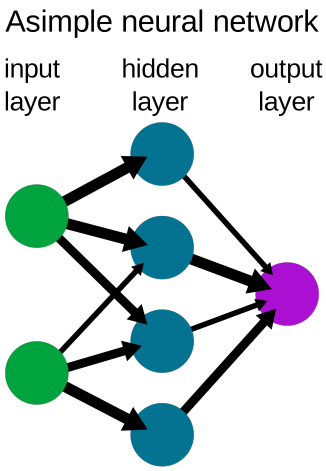

Figura 2.1: Ejemplo de una red neuronal simple (By User:Wiso - from en:Image:Neural network example.svg, vetorialization of en:Image:Neural network example.png, Public Domain)

Imaginemos que dadas unas imágenes de 400 píxeles de números en blanco y negro, queremos utilizar una red neuronal para determinar qué número está presente en la imagen. Un píxel es la unidad mínima de información en color en una imagen digital, una imagen está formada por píxeles (Blinn (2005)). La red neuronal ya se encuentra entrenada para identificar este tipo de imágenes y cuenta con una capa de entrada, una de salida y una capa oculta de 15 neuronas. En este caso la red neuronal contaría con una capa de entrada de 400 nodos o neuronas, cuyo valor correspondería al valor del pixel correspondiente. Posteriormente, cada nodo recorrería sus conexiones a la siguiente capa, en este caso a la capa oculta. Para determinar si las siguientes neuronas son excitadas o no. Se utilizaría la función correspondiente. Dicha función cuenta con unos valores numéricos utilizados para el cálculo del resultado final de la función, estos valores numéricos son específicos de cada conexión de la red neuronal y se denominan pesos. Una vez calculada la capa oculta, se realizaría el mismo proceso para la capa de salida. Dicha capa estaría formada por 10 nodos los cuales determinan la probabilidad de que la imagen se corresponda a cada uno de los 10 dígitos.

Como podemos observar el proceso es bastante dependiente del valor asignado a los pesos de cada una de las conexiones de la red. Dichos pesos son determinados al realizar el entrenamiento de la red neuronal. Hay distintas maneras de realizar un entrenamiento, pero el más común y el que sería probablemente utilizado en el ejemplo anterior constaría de una serie de imágenes de dígitos en blanco y negro, las cuáles han sido clasificadas anteriormente por una persona según el dígito que representan. Esta información es transferida al modelo, el cuál modifica sus pesos una serie de veces, intentando cada vez más acercarse al resultado correcto.

Una vez finalizado todo el proceso y entrenado el modelo, digamos que las imágenes que necesitamos identificar no se encuentran en blanco y negro, sino que son a color. ¿Cuál es la solución en este caso para conseguir que el modelo sea capaz de

identificar este tipo de imágenes? En vez de entrenar una red neuronal nueva desde el principio para que sea capaz de identificar dichas imágenes, sería posible realizar un *fine-tuning* del modelo para obtener la funcionalidad deseada. Este proceso precisaría de nuevos datos para volver a entrenar al modelo, en este caso imágenes de dígitos en distintos colores, clasificadas e identificadas previamente por una persona. El proceso de *fine-tuning* volvería a entrenar el modelo pre-entrenado, pero esta vez los pesos de la red neuronal ya contarían con un valor que solo debe de ser ajustado para que funcione bien con los nuevos datos. Una vez realizado el proceso, nuestro modelo sería capaz de identificar imágenes de dígitos tanto en color como en blanco y negro.

Una vez explicado todo esto, hemos de discutir por qué motivo decidimos adentrarnos en este campo durante nuestra investigación. El motivo es simple: el resultado que buscamos obtener del modelo es bastante específico, probablemente tanto como para que ningún modelo pre-entrenado nos aporte dicha funcionalidad. Por ello, era vital para nuestro proyecto conocer como realizar este proceso en un modelo de lenguaje natural, para poder adaptar el modelo como nos fuera necesario.

2.4. Generadores de historias

Posteriormente vamos a hablar de una serie de generadores de historias prominentes actualmente en el mercado. Mediante este estudio analizamos las distintas aplicaciones de generación de narrativa, describiendo su funcionamiento y la posición de nuestra aplicación en el mercado.

2.4.1. AIDungeon

AIDungeon[12] es una aplicación web, presente también como aplicación en Google Play y en la App Store. Este generador se especializa en el desarrollo de narrativa guiada por un usuario. Este establece distintos ajustes en formato de texto, guiando a la aplicación en la generación de la historia. Además, el usuario puede crear mundos narrativos con distintos parámetros y campañas o historias en dichos mundos y compartirlos con otros usuarios. Una vez creados el usuario comienza a interactuar con la historia mediante comandos simples en forma de texto, como por ejemplo: Yo digo: Que buen día hace, voy a ir a dar una vuelta. Posteriormente la inteligencia artificial continua la historia por el usuario, dando la sensación al usuario de encontrarse en ese mundo personificando a su personaje. En caso de que el usuario lo desee también puede pedir a la inteligencia artificial que rehaga parte de la historia, o incluso editar partes anteriores. AIDungeon utiliza una serie de modelos basados en GPT-3 y se encuentra en continuo desarrollo.

[12] `https://play.aidungeon.io/main/home`

2.4.2. NovelAI

NovelAI[13] permite la generación de una historia de una manera muy similar a la ofrecida por AIDungeon. Sin embargo, NovelAI cuenta con algunas características distintivas. Una de estas características es el llamado *Lorebook*, que el usuario puede utilizar como una especie de diccionario para que el modelo sea capaz de recordar eventos importantes pasados. Esta funcionalidad ofrece un gran potencial a la aplicación, ya que en otros generadores el usuario debe de acumular todos estos conceptos en un largo mensaje que es enviado en cada entrada al modelo. Sin embargo, en NovelAI este proceso se realiza de una forma mucho más cómoda para el usuario.

NovelAI ofrece una versión gratuita limitada de unas 100 generaciones al mes. También ofrece la posibilidad de pagar una suscripción que permite una cantidad ilimitada de generaciones, así como acceso a modelos más avanzados, más memoria (lo cuál permite que el modelo recuerde más detalles entre generación y generación) y generación de imágenes.

2.4.3. ToolBaz

ToolBaz[14] no se trata de una herramienta, sino de un conjunto de herramientas capaces de generar texto mediante inteligencia artificial. En esta aplicación web el usuario ha de introducir una entrada en forma de texto, la cual da lugar a una historia o cualquier otro tipo de contenido dependiendo de la herramienta seleccionada. Este tipo de página puede ser utilizada no solo para la generación de historias, sino que también para escribir un primer borrador sobre cualquier tipo de texto. Incluye opciones tanto para crear poemas, como para desarrollar encuestas centradas en un tema en concreto. También ha introducido recientemente una alternativa a ChatGPT.

2.4.4. Conclusiones de la búsqueda de generadores de historias

Como se puede observar, ninguna de las herramientas presentes en el mercado ofrece la funcionalidad que nuestro proyecto busca ofrecer. Herramientas de generación de texto basadas en lenguaje natural que son capaces de crear historias llevan mucho tiempo estando presentes en el mercado. Sin embargo, todas ellas precisan de una entrada comprensible en forma de texto para poder cumplir esta función. Nuestro programa busca resolver esta limitación, permitiendo la utilización de un mapa para poder obtener historias cohesionadas e interesantes acerca de los distintos territorios. A continuación pasamos a hablar sobre el desarrollo del proyecto.

[13] `https://novelai.net`
[14] `https://toolbaz.com`

Capítulo 3

Generación de historias a partir de un mapa personalizado

En este capítulo se abordará en detalle el extenso trabajo llevado a cabo durante el desarrollo de este proyecto. Con el objetivo de brindar un mayor entendimiento del proyecto, explicaremos en que consiste el proceso de generación de una historia. Como se puede ver en la figura 3.1, para empezar, se deberá generar un mapa con el generador escogido, como explicamos en la sección 3.1.4. En ese apartado se explica también la extracción de los datos del mapa, que serán almacenados en un grafo como se explica en el apartado 3.3. A continuación se usarán los datos para la generación de texto usando los modelos que escogimos. La selección de los modelos se puede encontrar en la sección 3.1.2. El siguiente paso consistirá de someter el texto generado a un sistema de puntuación que juzga la calidad del texto y decide si es válido o no. Se pueden encontrar detalles al respecto en la sección 3.4. Finalmente se juntarán diferentes salidas para formar el resultado.

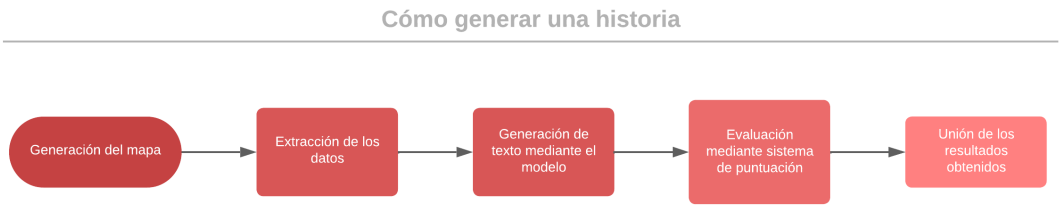

Figura 3.1: Proceso de generación de una historia de manera esquemática

Se comenzará por la selección de las tecnologías, que se basó en una evaluación de diversos pros y contras y la realización de pruebas y prototipos para determinar su eficacia. Se describirá el proceso de implementación del primer prototipo, donde se incorporaron dichas tecnologías al proyecto, abarcando tanto el modelo de lenguaje como el generador de mapas, así como la implementación de la lectura del mismo. Además, se detallará cómo se emplearon estas tecnologías para obtener los primeros resultados.

Posteriormente, se profundizará en el sistema de puntuación desarrollado, el cual fue diseñado con el objetivo de mejorar la calidad de los resultados y optimizar la salida de la herramienta. Se explicará el proceso de *fine-tuning*, así como la obtención de los casos utilizados para su re-entrenamiento y la metodología empleada para llevarlo a cabo de manera efectiva.

Por último, se presentará la evolución continua del proyecto hasta alcanzar el resultado actual. Se describirán las mejoras realizadas en la salida generada, detallando los ajustes y refinamientos implementados a lo largo del proceso de desarrollo.

3.1. Selección de Tecnologías

En este apartado describiremos el proceso de selección de tecnologías, incluyendo los modelos de generación de texto y los generadores de mapas contemplados para el proyecto.

3.1.1. Selección del entorno y del lenguaje

En primer lugar, tuvimos que elegir el lenguaje de programación más adecuado para nuestro proyecto. Los lenguajes disponibles en Google Colab son R, Julia, JavaScript y Python. R es un lenguaje de programación enfocado al análisis estadístico y la visualización de datos (Ripley et al. (2001)), por lo que fue descartado al no tener relación alguna con nuestra herramienta. Julia, por otro lado, es un lenguaje de programación de alto nivel y de código abierto diseñado específicamente para el cómputo científico, la computación numérica y el análisis de datos (Bezanson et al. (2017)). Los usos de este lenguaje tampoco están relacionados con nuestra herramienta. JavaScript, un lenguaje utilizado principalmente en el desarrollo web y la creación de aplicaciones interactivas en el navegador (Eguíluz Pérez (2012)). Sin embargo, no dispone del mismo catálogo de librerías que nos podría ofrecer Python respecto a procesamiento de texto. Por último, Python se trata de un lenguaje interpretado, con tipado dinámico, fuertemente tipado, multiplataforma y orientado a objetos (González Duque (2011)). Es un lenguaje bien conocido, que dispone de una comunidad muy activa y de una gran variedad de bibliotecas y herramientas útiles para el proyecto. En coclusión, acabamos decantándonos por el uso de Python como lenguaje de programación del entorno.

Dentro de los entornos que ofrecen la posibilidad de programar en Python, dos de ellos nos resultaron más atractivos debido a la facilidad que aportan para trabajar con modelos de lenguaje, estos son: Jupyter Notebook y Google Colab.

Jupyter Notebook es un proyecto de código abierto que brinda una plataforma computacional web y permite crear cuadernos interactivos. Estos cuadernos ofrecen la posibilidad de combinar código, elementos multimedia y texto explicativo en un solo entorno. Es compatible con varios lenguajes de programación, incluyendo Julia, Python y R, de los cuales proviene el nombre Ju-pyt-er (Díaz García y Cabrera Granado (2018)). El método más eficaz de usar Jupyter Notebook es creando cuadernos que contienen celdas donde puedes escribir código en los lenguajes mencionados anteriormente. Además, también puedes escribir en lenguaje Markdown para explicar

y documentar el código de una manera más cómoda. Para hacer uso de Jupyter Notebook en un entorno local o en un servidor, es necesario realizar una instalación y configuración adicional del software. Esto implica establecer las dependencias de los lenguajes de programación a utilizar, así como la configuración de los recursos adicionales (como el acceso a GPU o TPU).

Por otro lado, Google Colab o Colaboratory, es una plataforma web que permite la creación de cuadernos interactivos. Al igual que Jupyter Notebook, Google Colab ofrece la capacidad de combinar código y texto explicativo en un entorno colaborativo en línea. Es compatible con varios lenguajes de programación, incluyendo R, Julia, JavaScript y Python. A diferencia de Jupyter Notebook, Google Colab proporciona un entorno de ejecución en la nube ya preconfigurado. Lo que quiere decir que no es necesario realizar instalaciones o configuraciones adicionales. Adicionalmente, se pueden aprovechar recursos de hardware como GPU y TPU sin tener que configurarlos por separado (Bisong y Bisong (2019)). La simplificación del proceso de configuración evita las posibles complicaciones técnicas. Además, Google Colab está integrado con Google Drive, lo que permite almacenar y gestionar los cuadernos en la nube. Facilitando así el acceso a los cuadernos desde diferentes dispositivos y proporciona una opción de respaldo automático de los trabajos realizados.

Cabe destacar que tanto en Google Colab como en Jupyter Notebook es posible invitar a usuarios y colaborar en tiempo real. Sin embargo, hay diferencias en la forma de acceder a los cuadernos y colaborar con otros usuarios. Por un lado, que en Google Colab basta con enviar un enlace a los colaboradores para que puedan acceder al cuaderno a través de su navegador web con todas las configuraciones predefinidas. Por otro lado, en Jupyter Notebook es necesario tener la aplicación instalada localmente. Esto implica que si se desea trabajar en un ordenador que no tiene Jupyter Notebook instalado, será necesario realizar la instalación. Esta dependencia de instalación puede suponer un obstáculo y limitar el avance del trabajo si no se cuenta con acceso a un ordenador con la aplicación instalada.

Teniendo en cuenta estas consideraciones, la elección de Google Colab como nuestro entorno de trabajo nos ha proporcionado una solución conveniente y eficiente para la colaboración y el acceso a recursos de hardware. Además, no requiere de instalaciones adicionales o configuraciones del entorno, permitiéndonos avanzar de manera más fluida en el desarrollo.

3.1.2. Modelo de lenguaje

Como entrenar un modelo de lenguaje sería una tarea costosa y demandaría mucho tiempo, decidimos explorar alternativas y nos encontramos con HuggingFace[1], una plataforma de código abierto especializada en aplicaciones de procesamiento de lenguaje natural o NLP (Wolf et al. (2020)). HuggingFace nos resultó particularmente interesante por dos razones. En primer lugar, su biblioteca de Python llamada Transformers ofrece una amplia API que incluye los modelos que planeábamos utilizar. En segundo lugar, la plataforma cuenta con una sección llamada HuggingFace Model Hub, donde se encuentran disponibles diversos modelos de lenguaje pre-entrenados para distintas tareas de procesamiento de lenguaje natural.

[1] https://huggingface.co/

Como ya mencionamos en el Estado de la Cuestión, nunca habíamos trabajado con Transformers, por lo que tuvimos que recolectar toda la información necesaria para implementar el modelo en nuestro entorno, así como el uso de sus parámetros y maneras de explotar su potencial (Pita y Arribas (2021)) Durante nuestra investigación, encontramos modelos que se ajustaban a nuestros objetivos y resultaron ser muy útiles. Sin embargo, descartamos aquellos que requerían el pago de *tokens* o tenían limitaciones de uso, ya que no queríamos comprometernos con restricciones a largo plazo.

Encontramos un modelo llamado `gpt2-genre-story-generator`[2] que captó nuestra atención. Este modelo es capaz de generar historias breves a partir de una frase incompleta. Por ejemplo, si proporcionamos la frase "*Her name was Julian and...*", el modelo continuará generando un párrafo basado en esa frase. No hay una limitación en cuanto a la longitud de la frase proporcionada por el usuario, pero es importante destacar que a medida que aumenta la extensión de la entrada, la coherencia en la generación disminuye.

Una característica interesante de este modelo es que cuenta con una variable para categorizar el género de la historia a producir. Entre las opciones disponibles se encuentran: "Superheroe", "horror", "drama", "action", "thriller" y "sci-fi". Esta capacidad de adaptarse a diferentes géneros añade versatilidad al generador de historias.

Posteriormente vimos otro modelo similar al mencionado anteriormente, `gpt2-small-spanish`, pero desarrollado específicamente para el idioma español[3]. Este modelo funciona de manera similar en cuanto a su capacidad de generar historias en base a una frase inicial. Sin embargo, a diferencia del modelo anterior, este no ofrece una clasificación del género de las historias generadas. Una limitación importante de este modelo es que fue entrenado exclusivamente utilizando datos de la Wikipedia en español. Aunque la Wikipedia puede proporcionar una gran cantidad de información útil, no siempre pasa por un proceso de filtrado riguroso para eliminar contenido no confiable o inapropiado presente en internet. Debido a esta limitación, decidimos no emplear este modelo en nuestro proyecto.

3.1.2.1. gpt2-genre-story-generation

Encontramos otro modelo que se enfoca en la generación de pequeñas historias con una variedad de temas, como romance, aventura, misterio y detectives, fantasía, humor y comedia, paranormal y ciencia ficción. Este modelo nos permite generar historias en inglés sobre cualquiera de estos temas. Sin embargo, es importante tener en cuenta que tiene una limitación en cuanto a la longitud de las historias generadas, ya que tienden a perder coherencia rápidamente. Este modelo fue entrenado utilizando novelas gratuitas publicadas por autores anónimos que se recopilaron del conjunto de datos del BookCorpus[4]. Como resultado, las historias generadas tienen una estructura similar a la de las novelas y suelen estar escritas desde el punto de

[2] `https://huggingface.co/pranavpsv/gpt2-genre-story-generator?text=My+name+is+Thomas+and+my+main+quest+is+destroy+my+village+by+all+means+possible.+And+after+knowing+that+my+wife+Mary+and+their+little+daughter+Narmarka+will+die+in+the+hands+of+their+enemy+and+if+this+war+does+not+succeed+then+there+is+no`

[3] `https://huggingface.co/datificate/gpt2-small-spanish`

[4] `https://huggingface.co/datasets/bookcorpus`

vista de un narrador. Esta característica nos pareció interesante, ya que proporciona una experiencia de lectura más inmersiva y cercana a la narrativa de una novela. Este modelo puede encontrarse en HugginFace[5]

3.1.2.2. GPT-Neo-StoryGeneration

Por último, también nos interesó el modelo `GPT-Neo-StoryGeneration` . Aunque la página no proporcionaba mucha información detallada, realizamos varias pruebas de generación de texto y observamos que las salidas eran muy similares a lo que estábamos buscando. Además, al ser una variante de GPT-Neo, existe la posibilidad de realizar un *fine-tuning* en el futuro para adaptarlo aún más a nuestros objetivos específicos. Esta flexibilidad nos brinda la oportunidad de refinar y mejorar el rendimiento del modelo para obtener resultados más precisos y coherentes en nuestras tareas de generación de texto. Este modelo también puede encontrarse en HugginFace [6]

3.1.2.3. ChatGPT

Durante nuestro trabajo con los modelos, descubrimos ChatGPT, un *bot* conversacional desarrollado por OpenAI que utiliza GPT3.5. En ese momento, ChatGPT se encontraba en beta, pero posteriormente se lanzó de forma gratuita para entrenamiento. Encontramos algunos repositorios en GitHub y tutoriales que explicaban cómo integrar ChatGPT en un programa de Python. Sin embargo, nos encontramos con un problema al intentar implementarlo: debido a su gran popularidad, el servidor de ChatGPT experimentaba caídas frecuentes. Depender de su disponibilidad no era viable para nuestra aplicación, por lo que finalmente decidimos descartarlo. No obstante, consideramos utilizarlo en el futuro si se lanzara un modelo superior, ya que ChatGPT es uno de los mejores en el mercado hasta el momento. Una de las características destacadas de ChatGPT es su capacidad para retener las últimas 4000 palabras de la conversación, lo que permite generar respuestas coherentes y tener en cuenta el contexto previo. Esta característica sería muy útil para nuestro TFG, especialmente al crear historias con coherencia y sentido sin necesidad de verificar constantemente su cohesión.

3.1.2.4. Elección final del modelo

En resumen, consideramos que `gpt2-genre-story-generation` podía ser una opción prometedora para nuestro proyecto debido a su capacidad para generar historias en diferentes temas y su entrenamiento basado en novelas del *BookCorpus*. La fácil personalización de sus parámetros nos permite adaptar las salidas a nuestras necesidades. Sin embargo, debemos tener en cuenta la limitación en la longitud de las historias generadas y asegurarnos de que se mantenga la coherencia en la medida de lo posible.

Junto a ese modelo seleccionamos también el modelo `GPT-Neo-StoryGeneration` debido a que las salidas que generaba diversas pruebas que realizamos eran muy

[5] `https://huggingface.co/aspis/gpt2-genre-story-generation`
[6] `https://huggingface.co/PatrickTyBrown/GPT-Neo_StoryGeneration`

similares a lo que buscábamos en términos de calidad. Además, pensamos en la posibilidad de realizar un *fine-tuning* en el futuro. Esta capacidad de ajuste nos brinda la flexibilidad necesaria para adaptar el modelo de manera más precisa a nuestros objetivos específicos.

Una vez encontramos los dos modelos de HuggingFace y seleccionamos Google Colab como entorno de desarrollo para nuestro proyecto, comenzamos a considerar qué generador de mapas utilizar. Nuestro objetivo era crear un mapa sencillo con un par de regiones y un tamaño reducido. Luego, tendríamos que procesar e interpretar los datos más relevantes del mapa y comparar las respuestas generadas por ambos modelos. Una vez elegido el modelo que ofreciera las respuestas más cercanas a nuestras expectativas, nos enfocaríamos en refinar la salida y crear una herramienta útil para el mercado. Este proceso de pulido nos permitiría obtener resultados óptimos y brindar una solución efectiva a los usuarios.

3.1.3. Selección de librerías para el sistema de puntuación del texto

Como ya mencionamos anteriormente, queríamos hacer un sistema de puntuación que evaluara con una nota los textos generados por el modelo y así seleccionar el mejor de todos. Para hacer el sistema de puntuación trabajamos con las librerías de python que se describen a continuación:

difflib

difflib[7] es una biblioteca de python que se usa para la comparación de secuencias de datos, como en nuestro caso, cadenas de texto. Dentro de la librería encontramos una clase llamada *SequenceMatcher* que compara parejas de secuencias de cualquier tipo. Gracias a esa clase podemos mejorar el sistema de puntuación revisando las salidas generadas para ver si ocurre algún *Looping Behavior* explicado en la sección 2.2.0.2

nltk

Gracias a *Difflib* habíamos solucionado el *Looping Behavior*, pero nos seguíamos encontrando con *Word Looping*. Para erradicar ésto utilizamos nltk (Natural Languaje ToolKit)[8].

Se trata de una librería que trabaja con datos del lenguaje humano. Utilizamos nltk para poder Tokenizar. Tokenizar es un proceso en el que una entrada es transformada en tokens o unidades que el modelo puede entender para utilizar en su entrenamiento o procesamiento. Esta tokenización de los datos en algunos casos es realizada de manera automática por el modelo al introducir un prompt. Gracias a la *tokenización* podemos dividir el texto en unidades más pequeñas llamadas *tokens*. Con nltk *tokenizaremos* todas las palabras que se encuentren en la oración y

[7] `https://docs.python.org/3/library/difflib.html`
[8] `https://www.nltk.org`

nos devuelve una lista con todas las palabras encontradas. Así resultará mucho más rápido evaluar si la salida del modelo ha sufrido de *Word Looping*.

re

Para evitar la aparición de ciertas palabras también queríamos encontrar un método rápido de analizar frases en buscas de estas palabras. Para ello encontramos re (Regular Expression Operations)[9]. Es un módulo muy extenso que destaca por métodos para la búsqueda de elementos en una cadena de texto. Utilizamos esta librería para varias secciones del sistema de puntuación que mencionaremos más adelante, ya que nos servía tanto para detectar palabras no deseadas en el texto como para limpiar la oración eliminando espacios en blanco o apostrofes mal colocadas que hacía que la evaluación de sintaxis diera un falso negativo.

Pytorch

PyTorch es una librería de aprendizaje profundo altamente utilizado que proporciona herramientas y funcionalidades para construir, entrenar y desplegar modelos de redes neuronales. Se destaca por su enfoque flexible y su capacidad para realizar cálculos eficientes en GPUs, lo que lo hace ideal para proyectos de investigación y desarrollo de inteligencia artificial. El proceso de *fine-tuning* con PyTorch implica cargar un modelo pre-entrenado y congelar sus capas iniciales para preservar el conocimiento ya adquirido. Luego, se agregan capas adicionales al modelo para adaptarlo a la tarea específica. Más adelante hablaremos sobre todo el proceso de *fine-tune* y como hemos utilizado pytorch para conseguirlo.

Spacy

Por último nos faltaba lo más importante, la sintaxis y la cohesión de textos. Para resolver estos problemas usamos Spacy[10], una librería con una gran variedad de herramientas para analizar textos, comprobar si la sintaxis está bien, extraer información para comprobar la cohesión, etc. Para aplicar todos estos cambios usamos la *Lematización*, que consiste en reducir una palabra a su forma base o lema, de esta manera la comparación de palabras como adjetivos o verbos se hace más eficaz que si tuviéramos por ejemplo el verbo conjugado, evitando así más errores del sistema de puntuación.

3.1.4. Selección del generador de mapas

Como ya mencionamos anteriormente, seleccionar el mejor generador de mapas es una tarea extensa, ya que queríamos que tuviera varios requisitos mínimos para facilitarnos el trabajo y actualmente existen muchos generadores. Las cualidades deseadas, establecidas por nosotros, para poder darle uso a un mapa de manera óptima eran:

[9] `https://docs.python.org/3/library/re.html`
[10] `https://spacy.io/usage/spacy-101`

1. Que se pudiera exportar una vez creado (Preferiblemente en formatos que ya hemos trabajado como JSON)

2. Que siga recibiendo actualizaciones, pues al estar en constante mejora nos facilita así un servicio técnico y ayudas si nos desorientamos al usarlo

3. Que tenga múltiples opciones para modificar parámetros, pues así no tendríamos que cambiar de generador si decidimos ampliar el proyecto en un futuro y se nos queda corto

4. Que sea fácilmente modificable para el usuario para que no tenga que saber conocimientos de programación o similares para poder usar nuestro programa

Una vez establecidos los requisitos mínimos comenzamos a trabajar con los generadores ya investigados para decidir cual se aproximaba más a nuestras necesidades.

3.1.4.1. Donjon

A pesar de la amplia variedad de herramientas disponibles en la página[11] tales como creación de mapas de terrenos, mazmorras o generación de nombres de ciudades y ríos, nos encontramos con la limitación de que la información generada se presenta de forma fragmentada en diferentes formatos. Por un lado, se obtiene el mapa en sí, por otro, un archivo de texto plano con los nombres de los estados,y así. Esta dispersión de información no se ajusta a nuestros objetivos, ya que buscamos obtener un mapa con toda la información integrada. Además, nos encontramos con la dificultad de que el único formato de exportación disponible es en JPG o PNG, lo cual dificulta en gran medida la parametrización del mapa. Como resultado, decidimos descartar esta opción.

3.1.4.2. Inkarnate

Aunque este generador es ampliamente reconocido y cumple con muchos de nuestros requisitos[12], presenta ciertas limitaciones que nos llevaron a descartarlo como opción. Por un lado, cabe destacar su capacidad de personalización, que es bastante amplia, y su interfaz intuitiva, lo cual lo hace accesible incluso para aquellos sin conocimientos en diseño o programación. Además, el hecho de que continúen actualizando el software y solucionando problemas es beneficioso, ya que nos brinda la oportunidad de recibir ayuda y resolver cualquier inconveniente que puedan surgir. La existencia de una comunidad activa en torno a esta herramienta también es un punto a favor.

No obstante, hemos tomado la decisión de descartarla por dos razones fundamentales. En primer lugar, la capa de personalización se ve limitada en la versión gratuita, ya que para acceder a todas las características es necesario suscribirse a una versión Pro de pago, ya sea mensual o anual. Esto implica restricciones en la cantidad de mapas que se pueden crear, así como en la calidad de los mapas y la

[11] `https://donjon.bin.sh/`
[12] `https://inkarnate.com/`

cantidad disponibles. Por otro lado, el problema más importante radica en los formatos de exportación, ya que Inkarnate solo ofrece la posibilidad de exportar mapas en PNG y JPG. Esta limitación dificulta enormemente la lectura e interpretación de la información del mapa, ya que no podemos acceder a los datos subyacentes mediante una simple imagen. Por lo tanto, hemos decidido descartar también esta opción.

3.1.4.3. Wonderdraft

Wonderdraft es un generador altamente conocido en la comunidad[13] que se destaca por su enfoque en la creación de mapas con un estilo fantástico o épico. A diferencia de Inkarnate, este generador ofrece una mayor capacidad de personalización a nivel geográfico, permitiendo modificar tanto el relieve del terreno como su texturización. Sin embargo, esta mayor personalización también conlleva una curva de aprendizaje más pronunciada en comparación con otras herramientas, por lo que puede resultar algo complejo para un principiante manejarse en la página web.

A pesar de las ventajas mencionadas, encontramos los mismos problemas que con el resto de generadores. El generador solo ofrece la opción de exportar los mapas en formatos PNG y JPG, lo que limita nuestra capacidad para parametrizar los valores configurados durante la creación del mapa. Al depender únicamente de una imagen, perdemos gran cantidad de información subyacente. Además, debemos considerar el aspecto económico. Este generador, al ser más potente en términos de personalización, también tiene un precio más elevado. Según nuestra investigación, su costo es un pago único de 30 dólares. Dado este factor y las limitaciones mencionadas, hemos decidido descartar esta opción para nuestro proyecto.

3.1.4.4. Azgaar's Fantasy Map Generator

Finalmente encontramos el que sería el generador de mapas que utilizaríamos para nuestro proyecto[14], un generador de alta calidad con una cantidad de opciones y parámeteros personalizables inmensa. Éste generador es completamente gratuito y de código abierto, y de todos los que buscamos, era el único que cumplía todos los requisitos que mencionamos en el Estado de la Cuestión:

- Se puede exportar a JSON o geoJSON especificando si queremos un export más minimalista o uno completo con toda la información, haciendo mucho más asequible la parametrización de sus datos.

- Está en constante crecimiento, y cada mes están sacando una actualización, teniendo un soporte técnico activo que nos podría ayudar por si ocurriera algún inconveniente de algún tipo. Además, todas las actualizaciones que crean son compatibles con los archivos de mapas de versiones anteriores, por lo que no nos interfiere de ninguna manera que siga mejorando.

[13] `https://www.wonderdraft.net/`
[14] `https://azgaar.github.io/Fantasy-Map-Generator/`

- La comunidad que lo usa es tan grande que tienen hasta un Servidor de Discord de más de 2500 personas dónde hay moderadores, servicio de asistencia,un apartado dudas y ayudas para la gente principiante, y un largo etc

- Tiene incluso un exceso de contenido, ya que incluye toda la información geográfica del territorio, las civilizaciones que había, religiones inventadas, y hasta el nombre de cada uno de los habitantes del mundo.

- Pese a todo el contenido que tiene, posee una interfaz muy cómoda e intuitiva con una capacidad de personalización muy alta, por lo que resulta muy fácil cambiar el mapa y modelarlo a tu gusto, además de tener una opción donde puedes guardar un mapa, cargarlo en otro día y continuar modificándolo.

El formato JSON generado al exportar el mapa presenta una estructura de una sola línea que incluye todos los atributos del mapa. En el listado 3.1, se puede apreciar una sección del mapa en formato JSON, concretamente los estados. Dentro de la sección cada estado está separado por llaves, y dentro de cada uno se encuentran desde atributos importantes, como el nombre y la forma de gobierno, hasta atributos menos relevantes que hemos decidido omitir en la figura, como todos los nombres de los habitantes de cada estado.

Listing 3.1: Sección del mapa muy pequeña que muestra como se estructura la información de un estado


```
"states": [
{
   "i": 1,
   "name": "Oporica",
   "expansionism": 1.6,
   "capital": 1,
   "type": "Generic",
   "center": 3566,
   "culture": 12,
   "neighbors": [0, 3, 2],
   "diplomacy": [
   "x",
   "x",
   "Rival",
   "Rival",
   "Ally",
   "Suspicion",
   "Neutral",
   "Friendly",
   "Ally",
   "Friendly",
   "Ally",
   "Unknown"],
   "form": "Theocracy",
   "formName": "Theocracy",
   ...
```

```
    ...
}
```

En resumen, acabamos escogiendo a Azgaar's Fantasy Map Generator por ser el mejor generador de mapas que encontramos en el mercado completamente gratuito. A continuación hablaremos de cómo usar esta herramienta para crear tu propio mapa con los atributos personalizados.

3.2. Manual de uso del generador de mapas

El generador proporciona una amplia cantidad de información, permitiendo al usuario seleccionar qué atributos desea mostrar en la imagen resultante. La figura 3.2 ilustra un ejemplo en el cual, a partir de un solo mapa creado, se pueden generar infinitas representaciones visuales según las preferencias del usuario. Es posible mostrar diversos elementos, como ríos, estados y rutas, en una imagen 2D, o incluso generar una escena 3D que destaque biomas, emblemas y relieves. Esta flexibilidad brinda la oportunidad de visualizar el mapa de manera personalizada y adaptada a las necesidades del usuario.

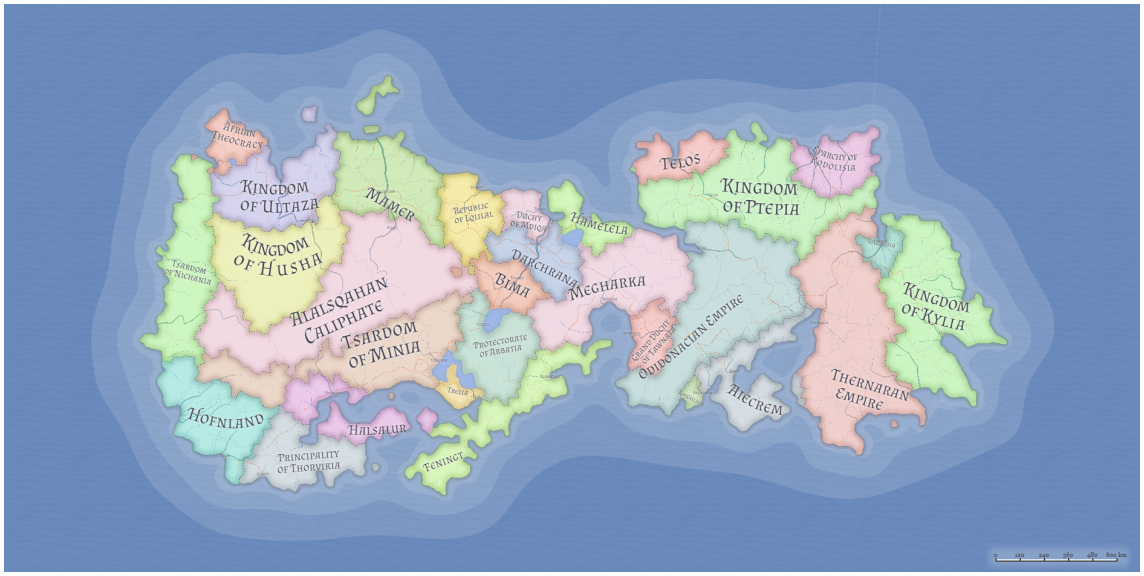

Figura 3.2: Mapa creado con Azgaar que visualiza rios, estados, hielo, barra de escala y rutas

La herramienta propuesta permite la instalación en el ordenador, lo que brinda la ventaja de poder trabajar sin conexión a Internet, a diferencia de utilizarla en un navegador. Al abrir la aplicación, se genera un mapa completamente aleatorio cada vez. La interfaz proporciona diversas opciones para modificar el mapa obtenido o generar uno nuevo con parámetros personalizados. En este trabajo se describirán resumidamente las cinco opciones principales de la herramienta.

- Opción 1: `Layers`: Esta opción no modifica el mapa en sí, sino que permite personalizar su visualización. Se pueden seleccionar diferentes tipos de mapas, como políticos, de biomas o de provincias, y ajustar los parámetros para

mostrar elementos específicos, como las celdas que dividen las regiones o la información de temperatura y precipitación.

- Opción 2: `Style`: Esta opción permite modificar la forma en que se visualizan los parámetros del mapa, sin cambiar sus valores. Es posible ajustar colores, opacidad, añadir filtros y otras configuraciones visuales. Además, se brinda la posibilidad de modificar directamente el código JSON que controla la visualización, para realizar ajustes más personalizados.

- Opción 3: `Options`: Aquí se pueden cambiar atributos del mapa, como el tamaño del lienzo en píxeles, el año y la era representados, el nombre del mapa y el número de culturas presentes. Sin embargo, estas opciones no se aplican al mapa actual, sino que se aplicarán al siguiente generado al pulsar la opción `New Map!`. También se proporcionan configuraciones adicionales para el propio generador para hacer más cómoda la modificación del mapa,así como el tamaño y color de la interfaz. Además, se incluye un botón crucial en la parte inferior llamado `Reset to default`, que permite eliminar todas las modificaciones realizadas a los parámetros y recargar la página con un mapa nuevo que presenta los valores por defecto.

- Opción 4: `Tools`: En esta sección, se pueden modificar atributos del mapa y acceder a información detallada sobre el mismo, como estados y su población, ríos y su longitud, entre otros. Destaca la opción "Mapa de altura", que permite ajustar la topografía del mapa. Sin embargo, se advierte tener precaución al modificar esta herramienta, ya que los elementos como ríos y ciudades dependen de la configuración predeterminada del mapa de altura. Se recomienda utilizar la opción "Borrar" para crear un nuevo mapa de altura personalizado y permitir que el sistema genere los parámetros secundarios nuevamente.

- Opción 5: `About`: Esta opción proporciona información acerca de la página y el proyecto en general. Además, se incluyen enlaces a video tutoriales, un manual de teclas de acceso rápido y una guía introductoria para ayudar a los usuarios a comenzar a utilizar la herramienta.

Aunque las opciones de la herramienta son intuitivas, también hay una línea de texto en la parte inferior de la página que brinda información adicional sobre cada elemento al colocar el cursor sobre ellos.

Finalmente, en la parte inferior de la interfaz, encontramos opciones para cargar, exportar, guardar y generar un mapa. Para que el mapa sea compatible con nuestro proyecto, debe ser exportado en formato JSON. Sin embargo, para visualizar el mapa en la aplicación, es necesario guardarlo previamente en formato .map, ya que el formato JSON no permite la visualización directa.

En resumen, el generador cuenta con una amplia gama de herramientas para personalizar el mapa según las preferencias del usuario, además de proporcionar información detallada y accesible que facilita el aprendizaje de su utilización, lo que lo convierte en una herramienta de fácil uso para cualquier persona interesada en utilizarla.

3.3. Procesado de datos y primer prototipo

Una vez encontramos el generador de mapas y seleccionamos los modelos de lenguaje, procedimos a realizar prototipos para evaluar la salida de ambos modelos y compararlos. Una vez generado el mapa y volcada su información en un fichero legible por nuestro programa, la elaboración de un simple prototipo nos permitiría probar los distintos modelos, así como establecer un punto de partida a partir del cuál ir iterando.

3.3.1. Generación del mapa

Para ello el primer paso del desarrollo consistía en la generación de varios mapas de distintos tamaños y densidad de culturas. En nuestro caso era importante contar con una casuística amplia a la hora de comparar los resultados obtenidos en la herramienta a desarrollar. Para el desarrollo inicial generamos dos mapas:

- Uno pequeño con una sola región, poca población y terreno limitado.
- Otro más extenso, con numerosas regiones, mayor población y una variedad de terrenos.

3.3.2. Volcado del mapa a JSON

Sin embargo, encontramos un problema en el proceso de visualización del mapa. Descubrimos que era posible exportar el mapa en formato JSON para su fácil lectura, pero no podíamos importarlo de vuelta en la aplicación para visualizarlo nuevamente. Para visualizar el mapa, era necesario contar con un archivo en formato .map, el cual debía ser guardado previamente en la página utilizando el botón "save" y eligiendo una de las tres opciones disponibles (Machine, Dropbox o Browser). Lamentablemente, nos dimos cuenta más adelante de que no podíamos cargar directamente el archivo JSON en la página y visualizarlo nuevamente. Por lo tanto, no podemos mostrar los mapas iniciales que utilizamos en las pruebas. Sin embargo, la descripción previa de cada uno de los mapas brinda una comprensión adecuada de su estructura y características. Es importante tener en cuenta este detalle para evitar el error en futuras implementaciones.

La estructura del archivo JSON se organiza de manera clara y ordenada, proporcionando una amplia cantidad de parámetros relevantes para nuestro trabajo. Al examinar el JSON, notamos que estaba compuesto por una única línea de código, pero contenía toda la información necesaria. Decidimos leer el JSON y extraerlo como un diccionario, lo cual nos permitió filtrar rápidamente la información irrelevante, como los nombres de los ciudadanos o los bloques de hielo del mapa.

Como se puede ver en el listado 3.2, se muestra un ejemplo de cómo funciona la estructura del JSON para la lectura en este caso de un estado y cómo se organizan los pares clave-valor:

Listing 3.2: Demostración de la estructura de datos del JSON para guardar información de un estado

```
{
   "i":0
   "name": "Bifjorian War",
   "start": 1226,
   "end": 1232,
   "diplomacy": ["x", "Ally", "Rival", "Friendly", "Neutral", "Neutral",
       "Unknown", "Suspicion", "x", "Suspicion", "Rival", "Friendly"],
   "form": "Monarchy",
   "formName": "Principality",
   "fullName": "Principality of Sildenesia"
   ...
}
```


Este JSON sigue la estructura de pares clave-valor, donde cada clave está seguida por su correspondiente valor. Para los valores simples, como el nombre del estado, se utiliza la clave "name" seguida por su valor "Bifjorian War". En el caso de valores más complejos, como las relaciones diplomáticas representadas por la clave "diplomacy", se utiliza una lista para enumerar las relaciones con otros estados. En resumen, la estructura del JSON sigue el patrón de clave-valor, donde se especifica una clave seguida de su valor correspondiente. Cada clave representa una propiedad específica y su valor contiene la información asociada a dicha propiedad.

Logramos organizar la información y guardarla en un diccionario que llamamos "data", lo cual nos permite acceder de manera sencilla a los datos de las celdas del mapa utilizando las claves correspondientes. Por ejemplo, si deseamos acceder a los estados del mapa, simplemente utilizamos `data[‘‘state’’]`. Esta estructura nos permite una extracción de datos más cómoda e intuitiva. Además, los parámetros se encuentran almacenados en relación con el estado al que pertenecen. Por ejemplo, al acceder a `data[‘‘religion’’][0]`, obtenemos la religión principal del primer estado. Esta característica facilita la manipulación y utilización de los datos según nuestras necesidades.

En los comentarios del código, hemos dejado registrados todos los parámetros que encontramos en el JSON. Además, hemos explicado aquellos parámetros que consideramos interesantes y que podrían ser utilizados en futuras etapas del proyecto. Aunque inicialmente no planeamos utilizar todos los datos del mapa, hemos decidido guardarlos, ya que podrían ser útiles en la generación de historias si decidimos ampliar el proyecto en el futuro.

3.3.3. Selección de material narrativo o *Story Sifting*

Para evaluar los parámetros y obtener las mejores salidas, se establecieron dos escenarios iniciales en el proceso de prueba. Se llevó a cabo una búsqueda de todos los estados del mapa para identificar un estado expansionista y otro no expansionista en el mapa. Si se cumplían ambas condiciones, se narraba una historia que involucraba únicamente los nombres de los dos estados seleccionados.

Esto nos permitía centrar nuestra atención en la evaluación y mejora de los modelos utilizando una entrada similar relativamente simple en todas las generaciones.

Sin embargo, durante esta etapa del desarrollo comenzamos de forma paralela a evaluar formas de posteriormente relacionar todos estos elementos narrativos para obtener una entrada mucho más compleja.

Una de las opciones consideradas consistía en dar la opción al usuario de centrar la generación de las historias en uno de los estados. Esto formaría historias relacionadas con este y ampliaría la historia desde este punto origen.

Sin embargo, finalmente optamos por otra opción. La utilización de un grafo de estados, donde cada estado se relaciona mediante sus lazos políticos, teniendo la opción posteriormente de relacionarlos mediante otros criterios. Esto nos permite generar varias historias por cada una de estas relaciones, teniendo en cuenta distintos parámetros como la densidad de población, religión, cultura...

3.3.4. Invocación del modelo de lenguaje

La entrada utilizada en las primeras generaciones para evaluar los modelos de lenguaje fue la siguiente:

```
Once upon a time in the world of "mapName", 2 cities,
"burgs[noExpansionist][name]" and "burgs[expansionist][name]"
got into a cruel war between each other.
```

Esta entrada era utilizada en el caso de encontrar un estado expansionista y otro no expansionista en el mapa. En caso de que no se encontraran dichos estados, la entrada utilizaba el nombre del mapa para contextualizar el relato de manera general:

```
Once upon a time in the world of "mapName", everyone was at peace.
In this prosperous world commerced started picking up.
```

Estas dos entradas nos permitían obtener resultados fácilmente comparables visualmente antes de dar una complejidad innecesaria en este punto a la entrada, entremezclandola con otros parámetros.

3.3.5. Configuración del modelo de lenguaje

Después de asegurarnos de que la lectura y el almacenamiento de los datos se realizaban correctamente, procedimos a instalar la biblioteca Transformers en nuestro proyecto. Esto nos permitió utilizar los modelos GPT-2 y GPT-Neo, realizar un prototipo inicial y evaluar las salidas generadas por estos modelos. Como mencionamos anteriormente en el apartado 3.1.2, seleccionamos dos modelos de la biblioteca HuggingFace que consideramos que podrían proporcionar los mejores resultados.

Dado que no teníamos experiencia previa en el manejo de modelos de lenguaje, realizamos una investigación exhaustiva sobre los parámetros más importantes que podíamos configurar y cómo usar el modelo[15]. Esto nos permitió comprender mejor cómo ajustar los parámetros para obtener resultados óptimos en nuestras tareas de

[15] `https://huggingface.co/blog/how-to-generate`

generación de texto. A continuación, se presentan los parámetros utilizados en el modelo de lenguaje durante el desarrollo del proyecto:

- Max Length: Representa el número máximo de *tokens* generados en la salida del modelo. Se estableció en 120 para equilibrar la longitud de las frases, evitando que fueran demasiado largas y perdiendo coherencia en el texto generado..
- Temperature: Controla el nivel de aleatoriedad y creatividad del modelo. Un valor de 0.1 genera frases más predecibles y repetitivas, mientras que un valor de 1 permite una mayor diversidad y imprevisibilidad en las salidas generadas. Se seleccionó un valor de 0.4 para mantener un equilibrio entre la coherencia del texto y la variabilidad en las frases generadas.
- Top p: Similar al parámetro de temperatura, el top p controla la diversidad de palabras consideradas en la distribución de probabilidad. Un valor de 0.95 se utilizó para fomentar el uso de palabras más diversas, aunque existe el riesgo de que el texto generado pueda carecer de coherencia en algunos casos. Valores más bajos resultaban en salidas repetitivas independientemente del número de ejecuciones.
- Repetition Penalty: Este parámetro gestiona la tendencia del modelo a repetir palabras o frases en la generación de texto. Oscila entre 1 y 2, donde 1 representa una penalización baja y 2 una penalización alta. En este caso, se utilizó un valor de 1.2, ya que los valores anteriores de los parámetros mencionados anteriormente reducían significativamente las repeticiones.
- top k: Top k: Determina el número máximo de palabras más probables que el modelo considerará en cada paso. Se mide en palabras y se estableció en 100, siendo un valor intermedio. Se recomienda utilizar valores entre 10 y 100 en modelos grandes como GPT-3, y entre 100 y 1000 en modelos más pequeños. Dado que se estaba utilizando GPT-2, se seleccionó un valor de 100, el cual proporcionaba resultados satisfactorios y una variedad adecuada en las salidas generadas.

3.3.6. Valoración de resultados

Al emplear ambos modelos, GPT-2 y GPT-Neo, se generaban dos salidas diferentes para cada uno, permitiendo así comparar y evaluar los resultados obtenidos. Dos ejemplos de las salidas iniciales que sacaron son:

- GPTNeo:

> Once upon a time in the world of Rinaroland 2 cities, Nesvolland and Nesvolland got into a cruel war between each other. The first city was located next to the river named Aetna; it had been built by King Villefort from a stone bridge over which he could not cross with his horse or cart for many years before the king's son was killed

> on that day when they were fighting together at Dordogne (the next city). He then took possession of the town from his father, and the king built a new castle of the same name, which he called "Nestor", or Nestil-Othos: Nespresis is a little settlement near here where there are also some small townsfolk who live along this road as well in the days that followed. In the days of the old kings Nescadocs lived among them, but the people of these parts died away after a long while

- GPT2:

> Once upon a time in the world of Rinaroland 2 cities, Nesvolland and Nesvolland got into a cruel war between each other.The first city was located next to the river named Theros. It had been centuries since their last battle at the end of the war that led them there. Afterward they fought until the city fell back with no hope for its survival or even if it did survive

Como podemos observar en los ejemplos anteriores, identificamos varias limitaciones en ambos modelos que no cumplían nuestras expectativas. Estas observaciones fueron las siguientes:

- Ambos modelos tendían a hacer un uso excesivo de entrecomillados para introducir frases en primera persona o diálogos, haciendo uso constante de expresiones como "dijo", "explicó" o "gritó". Esto dificultaba la creación de una narrativa fluida y natural, ya que nuestro objetivo era enfocarnos en los eventos que ocurrían entre los estados y los sucesos relevantes, en lugar de las historias individuales de los personajes.

- Los modelos hacían referencias frecuentes a eventos conocidos como la Segunda Guerra Mundial o lugares reales como Nueva York, lo cual carecía de sentido en el contexto de un mundo de fantasía creado por el propio usuario.

- GPT-Neo mostraba una tendencia a redundar en personas e historias narradas en primera persona, mientras que GPT-2 solía ofrecer descripciones más generales del entorno. Esto se debía a que el GPT-2 que utilizamos fue entrenado con una base de datos del Corpus de Novelas, compuestas por autores desconocidos, lo que propiciaba que se inclinara más hacia narradores impersonales en lugar de utilizar la primera persona.

El resto de salidas más relevantes que contrastan esta información se encuentran en el apéndice A

En vista de estas observaciones, decidimos continuar el desarrollo de la herramienta utilizando el modelo GPT-2. No obstante, también consideramos la posibilidad de entrenar el modelo GPT-Neo con conjuntos de datos diferentes para evaluar si su rendimiento mejoraba en comparación con GPT-2. Sin embargo, esto es un trabajo que abordaremos en etapas posteriores del proyecto.

3.4. Segundo Prototipo: Sistema de puntuación

A continuación, nos enfocamos en mejorar la entrada mediante la relación de los distintos estados, así como en el desarrollo de un sistema de puntuación que evaluara las salidas del modelo y asignara una nota para seleccionar la mejor frase generada.

3.4.1. Re-elaboración del material de entrada como grafo

Para facilitar la creación de la herramienta decidimos desarrollar en este momento la idea que mencionamos previamente de almacenar los datos en un grafo, en el cual los estados eran los nodos, que a su vez poseen todos los datos relacionados a este, y están unidos entre sí por aristas que indican la relación entre los estados, es decir, si son aliados, enemigos, amigos, desconocidos... Como se puede observar en la figura 3.3, los nombres de los estados son nodos, y las aristas están categorizadas por colores: azul si los estados son aliados, verde si son amistosos pero no aliados y rojos si son estados rivales. Solo mostramos esos en el grafo porque son los más relevantes, pero también pueden ser neutrales o desconocidos. De esta manera teníamos una lectura del mapa con todos los datos que podíamos obtener y posteriormente añadíamos al grafo solo los que fuéramos a usar para la herramienta, así si en un futuro queremos complicarlo solo tenemos que añadir más datos ya extraídos al grafo y meterlos en la entrada del modelo.

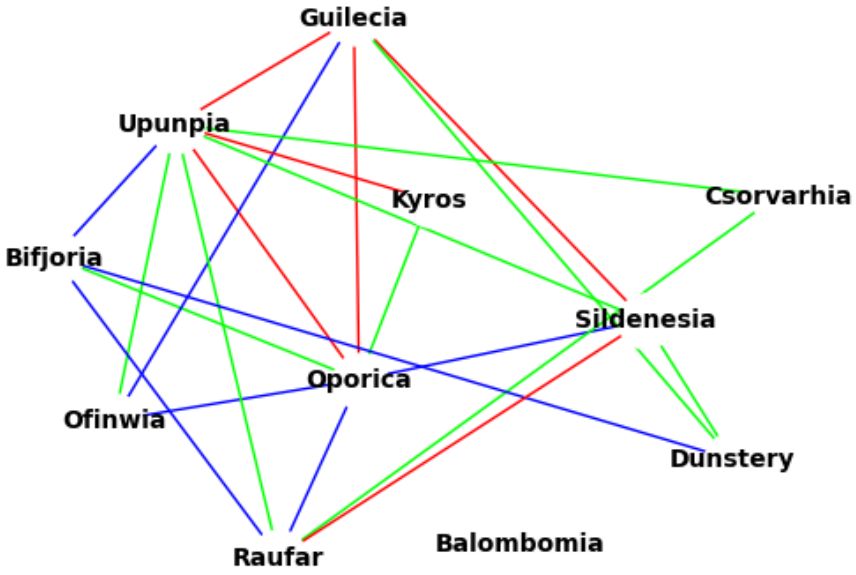

Figura 3.3: Grafo de un mapa pequeño con las relaciones entre todos los estados

3.4.2. Mejora del procedimiento de story sifting

La implementación explicada previamente nos permitía realizar la ampliación que estuvimos planificando previamente para el *story sifting*. En este primer modelo de la utilización del grafo seleccionamos un estado aleatorio de todo el grafo y realizamos un recorrido de todos sus estados vecinos. En el caso de alguno de ellos sean aliados o enemigos, se escribía una salida tratando sus relaciones.

Esta implementación nos permitía elaborar más sobre la entrada anterior en la que solo se obtenían dos estados, dando la posibilidad de ampliar la entrada y correlacionar varios estados entre sí. Además marcaba otro importante paso en nuestra meta de generar una historia que englobara a todos los moradores del mapa.

Sin embargo, esta funcionalidad no fue puesta en uso inmediatamente después a falta de mejoras sustanciales en la salida generada por los modelos.

3.4.3. Refinamiento del proceso de generación

Con las entradas del modelo mejoradas comenzamos con un sistema que llamaba 10 veces al modelo, les ponía una nota a cada salida y mostraba por pantalla el que mejor puntuación ha tenido de los 10. A menor nota se tuviera mejor era la salida, por lo que el objetivo del sistema sería sacar un cero. Todo el sistema se realizó gracias a las librerías y modelos de python mencionados en la sección 3.1.3. El criterio que creamos para evaluar las frases era:

- Repetición de frases: A diferencia de su sucesor GPT3, el *Word Looping* y *Looping Behavior* con GPT2 es más frecuente. En vez de poner una mala nota si encuentra una frase repetida lo que hacemos es ir sumando una puntuación por cada vez que detecte una frase repetida. Para ello utilizamos la libreria difflib y repasamos la salida comparando frase con frase utilizando una clase llamada `SequenceMatcher`. A esta clase como parámetros le pasamos las dos frases a comparar, y después llamamos al método `ratio()` que devuelve el porcentaje de coincidencias entre las dos frases, y si supera una coincidencia de 60 % entonces se considerará una frase repetida.

- Corrección de diálogos: Nuestro objetivo es generar una narración de los eventos o la historia del estado sin entrar en detalles sobre personas específicas. Para lograr esto, utilizamos el módulo *re* para localizar cualquier indicio de diálogo en primera persona, como palabras como "dijo", "habló", "susurró", pronombres o frases entre comillas que denoten la primera persona. Este proceso de corrección se evalúa mediante una puntuación que aumenta a medida que se detectan más errores en los diálogos generados.

- Corrección de hechos reales: Dado que nuestro mapa no se basa en el mundo real, es fundamental eliminar cualquier referencia a hechos históricos o lugares conocidos en el texto generado. Para lograr esto, creamos un diccionario que contiene los eventos históricos más relevantes, como la Primera Guerra Mundial, así como lugares altamente reconocidos, como Nueva York. Utilizando el

módulo *re*, filtramos el texto en busca de cualquier coincidencia con los elementos de nuestro diccionario y asignamos una puntuación adicional por cada evento o lugar mencionado.

- Errores sintácticos: Mediante el uso de *Spacy*, realizamos una evaluación de la corrección gramatical de las frases generadas. Nuestro objetivo es evitar la conjugación incorrecta de los verbos o la mezcla de narraciones en singular y plural, entre otros errores sintácticos. En caso de que la redacción sea adecuada, no se suma ningún punto adicional. Sin embargo, si se detectan errores sintácticos, se suma una puntuación única (aunque menor que en los casos anteriores, ya que priorizamos de más importante la repetición de frases y corrección del diálogo). Además, realizamos una corrección de los apóstrofos en el texto generado, ya que las salidas del modelo en inglés presentaban un mal uso del apóstrofo, usando ´ en vez de ', por lo que lo reemplazamos correctamente. Esta corrección nos permite evitar falsos negativos que podrían indicar incorrectamente que la frase está mal redactada.

- Coherencia de eventos: Esta sección del sistema es la más compleja de todas. Se enfoca en garantizar la coherencia de los eventos narrados. Para lograrlo, se eliminan las palabras superfluas tales como determinantes o artículos, y se realiza una lematización. La lematización consiste en una técnica de normalización de texto que busca reducir las palabras a su raíz o lema (KeepCoding (2023)). Posteriormente, se lleva a cabo una búsqueda de entidades en dos salidas del modelo y se puntúa en función de la coherencia de los eventos entre ellas. Por ejemplo, si una frase menciona que "El pueblo X está mejorando" y otra frase indica que "El pueblo X está empeorando", el sistema identificará la entidad "El pueblo X" y analizará si el evento es coherente entre ambas frases. *Spacy* es capaz de detectar adjetivos y categorizarlos como sinónimos o antónimos entre sí. Esto permite realizar una comprobación superficial, a través de frases simples, para determinar si dos oraciones son coherentes o no. En caso de que las oraciones carezcan de cohesión, se asignará una puntuación correspondiente al grado de redacción incorrecta. Es importante destacar que Spacy no es capaz de detectar de manera perfecta adjetivos contrarios o similares en todos los casos, lo que puede limitar la eficacia de esta sección del sistema en ocasiones.

La evaluación de la coherencia de eventos resultó ser más compleja de lo esperado, especialmente al comparar oraciones más complejas y analizar la coherencia entre sus sujetos, adjetivos y verbos. Aunque *Spacy* proporcionaba diversas herramientas, la tarea de comparación se volvía difícil. Por esta razón, decidimos posponer este criterio para futuras investigaciones y no incluirlo en la implementación actual. En cambio, nos enfocamos en mejorar el sistema de puntuación básico y la entrada del modelo. Observamos una notable mejora al utilizar un prompt básico de una frase. Sin embargo, nos dimos cuenta de que teníamos pocas entradas y poca información del mapa, ya que hasta ese momento el prototipo solo consideraba un estado aleatorio y exploraba sus estados vecinos para generar una salida basada en la presencia de estados aliados o enemigos. Para abordar esta limitación, decidimos aumentar la

longitud de la entrada del modelo y agregar más parámetros obtenidos del mapa, como la religión o el tamaño del estado. Esto nos permitió mejorar la entrada al tiempo que avanzábamos en el siguiente aspecto de desarrollo.

3.5. Tercer Prototipo: *Fine-tuning* de GPT-Neo

Los resultados que obtuvimos fueron muy diferentes al texto narrativo que queríamos mostrar. El resultado mostraba demasiados elementos no deseados, como lenguaje en primera persona, mención de eventos históricos, etc, como ya hemos mencionado anteriormente. Sin embargo, las herramientas desarrolladas posteriormente para evitar dichas ocurrencias solo nos permitían trabajar con el texto obtenido de los propios modelos, texto que casi siempre era generado con esos elementos indeseados. Simplemente pedir varias salidas de los modelos nos podía ayudar para minimizar dichos defectos, sin embargo, el tiempo de ejecución no paraba de ascender y no era una solución para nada definitiva.

Por ello, decidimos optar por otro método que ya investigamos con anterioridad, el *fine-tuning*. Dicho mecanismo nos permitía volver a modificar los pesos de los modelos pre-entrenados, aportándonos la capacidad de modificar la salida generada por el modelo, y así eliminar el problema de raíz.

El *fine-tuning* de un modelo de generación de lenguaje natural no fue una tarea sencilla y nos encontramos con diversos problemas.

3.5.1. Obtención de la base de datos

Como hemos explicado previamente, el *fine-tuning* de un modelo precisa de una serie de datos que posteriormente serán utilizados en el entrenamiento de la red neuronal. Dicha base de datos debe de cumplir con una serie de requisitos:

- La base de datos debe de contar con una cantidad significativa de datos. Dicho número depende de la cantidad de datos con los que haya sido entrenado previamente el modelo.

- Los datos deben de estar clasificados. Como ya dijimos anteriormente, la red neuronal debe de conocer durante su entrenamiento cuál es la salida esperada para cada dato de entrenada. Por lo que cada uno de los datos introducidos debe de contar además, con dicha salida.

- Los datos deben de poder ser tokenizados, es decir, poder ser convertidos a un formato que la red neuronal sea capaz de entender. Por ejemplo, en el caso del lenguaje natural, las palabras son entendidas como números. Por lo que es necesario poder obtener dichos números a partir del texto.

- Tanto la entrada como la salida de los datos debía de ser texto narrado, donde hubiera la mínima cantidad posible de elementos indeseados. Además, para que la generación concordara con la funcionalidad ya presente, la entrada debía de estar presente al inicio de la salida. De esta manera conseguiríamos que se

conservara el comportamiento del modelo que causaba que continuara con la entrada anterior en vez de generar texto independiente.

- Preferiblemente, la base de datos debía de narrar acontecimientos ficticios, pero la prioridad era que se centraran en la narración a gran escala de la civilización y sobretodo no en individuos.

Buscamos distintas bases de datos, intentando hallar aquella que cumpliera con todos los requisitos. Encontramos bases de datos con bastante fiabilidad[16], fruto de investigaciones y algunas incluso respaldadas por libros físicos recopilatorios. Sin embargo, no encontramos ninguna que nos sirviera para lo que estábamos buscando. La mayoría de bases de datos estaban centradas en la clasificación de texto, no en la generación de más texto a partir de otro. Además, las bases de datos o páginas recopilatorias de conflictos o sucesos entre civilizaciones no resultaron fácilmente extraibles, requiriendo una gran cantidad de tiempo y trabajo manual.

Por todo esto, optamos por utilizar una página web que recapitulara conflictos entre civilizaciones o naciones. En nuestro caso utilizamos una web [17] que recapitula una serie de conflictos históricos ocurridos entre los años 1800 y 1999 incluidos. Esta base de datos casera contaba con algunos elementos no deseados como acontecimientos históricos reales y en ocasiones citas, pero al menos se acercaba más al tipo de narración que buscamos. Todos los datos fueron extraídos manualmente a un archivo de formato JSON y fue organizado de la siguiente manera: los inicios de las narraciones fueron agrupados en una lista de elementos (array), y la narración entera fue almacenada en otra lista. De esta manera el elemento iésimo de la primera lista correspondía con el inicio del elemento iésimo de la segunda lista. La primera lista sería utilizada posteriormente como la entrada del dato, y la segunda lista como la salida esperada para ese dato por parte de la red neuronal. Ambas listas contaban con aproximadamente 1000 elementos.

3.5.2. Tokenización de los datos

Una vez obtenida la base de datos, debíamos de tokenizar los datos, convirtiendo las palabras en números que la base de datos pudiera comprender. Gracias a la investigación realizada previamente, conseguimos realizar dicho proceso de forma satisfactoria. Primero abrimos el fichero donde almacenamos la base datos. Posteriormente obtenemos las listas correspondientes a las entradas y a las salidas y las separamos en dos conjuntos desiguales. El 80 por ciento de las entradas y salidas a un conjunto para realizar el entrenamiento, y el 20 por ciento en otro para valorar el resultado obtenido posteriormente. Codificamos ambos conjuntos con el tokenizador del modelo de GPTNeo, asegurándonos de que los datos se encuentren en el mismo formato que el modelo. Además, definimos una clase Dataset de la librería Pytorch. Dicha clase es la utilizada por Pytorch para almacenar y procesar datos. Además, redefinimos los métodos init, len y getitem de la clase para que la implementación sea capaz de obtener los datos de la lista.

[16] `https://infoguides.gmu.edu/conflict/data` y `http://www.icb.umd.edu/dataviewer/`
[17] `http://www.onwar.com/index.html`

3.5.3. Entrenamiento del modelo pre-entrenado

Una vez preparados los datos, definimos algunas estructuras utilizadas por el entrenador del modelo. En primer lugar los parámetros del entrenamiento[18]:

- outputDir: define el directorio donde se escribirán las predicciones.
- numTrainEpochs: el número de *epochs* que debe de realizar el entrenamiento antes de finalizar. Un *epoch* es una unidad de medida de tiempo utilizada por sistemas informáticos. Utilizamos 3 epochs.
- perDeviceTrainBatchSize: el tamaño del *batch* de entrenamiento por cada núcleo de GPU/TPU/CPU. Establecido a 4.
- perDeviceEvalBatchSize: el tamaño del *batch* de evaluación por cada núcleo de GPU/TPU/CPU. Establecido a 4.
- learningRate: el ratio de aprendizaje de la clase AdamW de la librería Transformers. Establecido a 5e-5 el cuál es el valor por defecto.
- warmupSteps: número de pasos usados en el calentamiento lineal. Establecido a 500.
- saveTotalLimit: limita la cantidad de puntos de control que pueden permanecer en disco, borrando el más antiguo al crear uno nuevo. Establecido a 2 para ahorrar memoria en Google Colab.
- predictionLossOnly: devuelve únicamente el fallo al realizar la evaluación y generar predicciones. Establecido a verdadero.
- loggingDir: define el directorio donde se guardará el registro de las operaciones realizadas.
- loggingSteps: número de actualizaciones entre dos logs. Establecido a 500.
- evaluationStrategy: la estrategia de evaluación a seguir durante el entrenamiento, puede ser: no, steps o epoch. Lo establecemos a steps para que evalue después de evalSteps veces.
- evalSteps: número de actualizaciones entre dos evaluaciones si la estrategia de evaluación está configurada como steps. Establecido a 500 actualizaciones entre evaluación en nuestro caso.
- saveSteps: número de actualizaciones por steps entre dos checkpoints. Establecido a 500, de esta forma realiza un checkpoint al mismo tiempo que la evaluación.

Una vez establecidos todos estos parámetros, establecemos los parámetros del entrenador de Transformers y realizamos el entrenamiento y la evaluación, cada uno con su conjunto de datos correspondiente.

[18] `https://huggingface.co/transformers/v4.3.3/_modules/transformers/training_args.html`

3.5.4. Exportación/Importación del modelo

Una vez realizado el *fine-tuning* guardamos el modelo resultante en una carpeta compartida en Google Drive. De esta manera simplemente puede importarse el modelo resultante en vez de realizar de nuevo el entrenamiento. Para cargar el contenido montamos nuestra unidad de Google Drive en la sesión de Colab y accedemos a la carpeta compartida. Desde dicha carpeta, importamos el tokenizador y modelo resultantes del *fine-tuning*.

3.5.5. Utilización del modelo

Una vez importado el modelo de forma correcta procedimos a su implementación junto al resto de modelos. Dicho proceso fue rápido y solo tuvimos que acostumbrarnos a codificar y decodificar los datos cada vez que queríamos utilizar el modelo. A diferencia de los modelos pre-entrenados que utilizábamos previamente, en este modelo precisábamos de emplear una codificación manual de los datos utilizando el tokenizador resultante del proceso. Una vez aplicado dicho proceso una vez, el fragmento de código fue fácilmente replicable para cualquier instancia de uso.

3.6. Cuarto Prototipo: Mejora de la salida

Simultáneamente al desarrollo del *fine-tuning*, también nos dedicamos a mejorar el código que genera el prompt con el objetivo de agregar profundidad y aumentar la cantidad de información en la salida generada.

Para lograr una historia más detallada en la salida, en lugar de generar una única entrada y obtener una salida, decidimos incrementar el número de entradas para construir un párrafo completo con toda la información relevante sobre el estado. Esto se realiza teniendo en cuenta los siguientes aspectos:

1. Forma de gobierno del estado: Consideramos el tipo de gobierno del estado, como una monarquía o una república, y generamos una entrada específica relacionada con ello.

2. Religión del estado: Identificamos la religión predominante en el estado y la deidad principal a la que se rinde culto. Luego, generamos una entrada que refleje esta característica religiosa. También genera una salida si no tiene ninguna religión predominante.

3. Nivel de desarrollo del estado: Evaluamos si el estado está bien desarrollado o en progreso. Tomamos en cuenta la cantidad de territorio controlado por ciudades y generamos una entrada adecuada según la situación.

4. Tendencia expansionista del estado: Verificamos si el estado tiene una política de expansión territorial. Si es así, generamos una entrada que refleje esta ambición de expansión. En caso contrario, generamos una entrada diferente.

Al considerar estos aspectos en el código que genera el prompt, buscamos enriquecer la salida del modelo con información detallada y coherente sobre el estado en cuestión. En el proceso de generación de salidas, decidimos probar dos versiones diferentes para comparar sus resultados:

En la primera versión del proyecto, generamos las entradas mencionadas anteriormente por separado y luego las concatenamos al final para formar una única cadena de texto.

En la segunda versión, al generar una nueva salida, se incluye la salida generada anteriormente como parte de la entrada correspondiente. De esta manera, el modelo tiene en cuenta la información previa para generar una salida más coherente y conectada con la historia.

Capítulo 4

Resultados

Al final de nuestro proyecto obtuvimos una serie de resultados respecto a las mejoras y desarrollo implementado. En este capítulo vamos a describir los resultados obtenidos y cómo estos han influenciado al desarrollo de la versión final del proyecto.

En primer lugar se muestran a continuación dos ejemplos de nuestro punto de partida, es decir, salidas generadas por GPT2 y GPT Neo sin ningún tipo de *fine-tuning*, corrección sintáctica, valoración, ni eliminación de palabras:

```
GPT-Neo: "Once upon a time in the world of Rinaroland 2 cities,
Nesvolland and Nesvolland got into a cruel war between each other.
The first battle was fought onthe banksof Lake Trelle-tolte; the
second one in which the two men werefighting at once against their
enemy's forces under a greatfierce struggle that lasted for some
years until theywere finally routed by an army of Nelskorn who had
marched from Utterling toSweden with his father in the forest
behind him as he went along among themon foot after leaving home
the last night on this road.The first day of the third month of
the year, Neliswold came up again and met her husband from the
shoreup near Hrothgarweg (a town where there is no natural border)
who lived about three miles away.She saw the young man coming out
towards her, but she did not knowhim or understand what she"
```

```
GPT-2: "Once upon a time in the world of Rinaroland 2 cities,
Nesvolland and Nesvolland got into a cruel war between each other.
As King Ogh had been so often the one taking command...there was
no place for them either.  "No matter how many times I tried to
leave him there..." Aimee heard him squirm as she walked back
toward her hut. The doorbell rang in his ear just before him on
either side of the bed. She turned, and ran inside thinking that
it might be the end here.  At any rate he didn\'t have much choice
if Lyrlan came over and demanded his own food or what. He hadn\'t
eaten himself since coming home.  \'Goodnight Marnia!\' he said in
a low voice, a soft tone, that Kaelic royalty would be pleased at
seeing them through a distance. He didn\'t like going too far'"
```

En estos ejemplos el texto introducido en los modelos fue:

"Once upon a time in the world of Rinaroland 2 cities, Nesvolland and Nesvolland got into a cruel war between each other."

Ahora comparemos dicho resultado con la implementación de nuestra primera versión, la cuál describimos en 3.6. En esta implementación introducimos una serie de entradas por separado al modelo, cada una centrada en un aspecto específico de la civilización, como por ejemplo, su religión. Posteriormente concatenamos dichas salidas en una misma frase ofreciendo los siguientes resultados. Además, implementa todas las mejoras descritas previamente, además de un *fine-tuning* del modelo GPT Neo:

GPT Neo Fine-tuned: <BOS> <Fantasy> Some summers ago, Bifjoria instated a new form of government, Monarchy, and so far... <BOS> <Fantasy> The biggest known religion in the state was Norse Pantheon. A religion that was Polytheism, and venerated Thorfi, The Yellow Summer. It was a belief in the gods of the gods, and was a belief in the gods of the people. It was a belief in the gods of the people, and was a belief in the gods of the people. It was a belief in the gods of the people, and was a belief in the gods of the people. It was a belief in the gods of the people, and was a belief in the gods of the people. It was a belief in the gods of the people, and was a belief in the gods of the people. It was a belief in the gods of the people, and was a belief in the gods of the people. It was a belief in the gods of the people, and was a belief in the gods of the people. It was a belief in the gods of the people, and was a belief in the gods of the people.<BOS> <Fantasy> Some summers ago, a urban state was slowly setting it's rootsin the woods. The trees were low and the air was cool. A few miles away, a man was walking on a trail. He was a tall, lean, black-haired man with a long, dark beard. His face was a mask of a man's. He had a long, dark beard, and a mustache that was thick and dark. He wore a gray shirt and a gray coat. He had a long, dark beard, and a mustache that was thick and dark. He had a long, dark beard, and a mustache that was thick and dark. He had a long, dark beard, and a mustache that was thick and dark. He had a long, dark beard, and a mustache that was thick and dark. He had a long, dark beard, and a mustache that was thick and dark. He had a long, dark beard, and a mustache that was thick and dark.<BOS> <Fantasy> Long ago this state was struggling from it's lack of resources. However, the unity of it's people and the spirit of conquest led it to move forwardIt was a time when the people of the land were united in a common spirit. The people of the land were united in a spirit of peace and prosperity. They were united in a spirit of peace and prosperity.The people of the land were united in a spirit of peace and prosperity. They were united in a spirit of peace and prosperity.They were united in a spirit of peace and prosperity.They were united in a spirit of peace and prosperity.They were united in a spirit of peace and prosperity.They were united in a

```
spirit of peace and prosperity.They were united in a spirit of peace and
prosperity.They were united in a spirit of peace and prosperity.They were
united in a spirit of peace and prosperity.They were united in a spirit
of peace and prosperity.They were united in a spirit of peace and
prosperity.They were united in a spirit of peace and prosperity.They were
united in a spirit of peace and prosperity.
```

```
GPT-2: ""The people of this city are very happy with their life,"
said a priest named Niki.  "They don't like to be around strangers.
<BOS> <Fantasy> Some summers ago, a urban state was slowly setting
it's rootsSome summers ago, a urban state was slowly setting it's
roots in the west.  "I don't know what you mean," he said. " Leaving
this place is not going to happen." He looked at his watch and shook
his head. "We need to get out of here before we're too late for that."
The old man turned around and walked over to him.<BOS> <Fantasy> This
state was not very well developed, but it was happy with it's lot.
Its people a friendly bunch that were happy to seek peace with it's
neighboursThis state was not very well developed, but it was happy
with it's lot. Its people a friendly bunch that were happy to seek
peace with it's neighbours and the world as they knew it would be good
for them if they could get their hands on it.  "I'm sorry I don't want
you here."   She took her hand in hers and kissed him deeply.  "You are
my friend.
```

La primera versión tiene sus ventajas e inconvenientes. Por un lado, al tener entradas más simples, se generan salidas más limpias y con menos probabilidades de errores o incoherencias. Sin embargo, como se puede distinguir en el ejemplo, la principal desventaja es que no existe una coherencia directa entre cada salida. Ya que cada una es generada de forma independiente y no se tiene en cuenta la salida anterior.

Como podemos observar, las soluciones implementadas no consiguen eliminar por completo los comportamientos indeseados de los modelos. Sin embargo, si que consiguen obtener un mejor resultado, evitando los diálogos en primera persona en su mayor parte y sin símbolos residuales de la generación.

El modelo GPT Neo, sin embargo, se ha vuelto más propenso al *word looping* tras el proceso de entrenamiento. Esto se debe a la falta de un proceso de refinado de los datos presentes en la base de datos que creamos para realizar el entrenamiento, además de su reducida extensión.

Por último aquí incluimos unos ejemplos de la implementación de la segunda versión, donde las salidas se van acumulando en la siguiente entrada hasta formar un resultado de gran extensión:

```
GPT Neo Fine-tuned Salidas acumuladas: <BOS> <Fantasy> <BOS>
<Fantasy> <BOS> <Fantasy> <BOS> <Fantasy> Some summers ago,
Csorvarhia instated a new form of government, Monarchy, and
so far...  The biggest known religion in the state was Vengrian
Deities. The religion form was Polytheism, and venerated
```

Balasztes, The Worried Father of Fire. It was a religion of the most powerful people in the state, and it was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state, and it was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. In days gone by, this urban state was struggling to take hold and suffered from a long stagnationIt was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. It was the most powerful of all the religions in the state. This state was not very well developed, but it was happy with it's lot. Its people a friendly bunch that were happy to seek peace with it's neighboursThey were happy to see the state's people being happy to be with it's people. They were happy to see the state's people being happy to be with it's people. They were happy to see the state's people being happy to be with it's people. They were happy to see the state's people being happy to be with it's people. They were happy to see the state's people being happy to be with it's people. They were happy to see the state's people being happy to be with it's people. They were happy to see the state's people being happy to be with it's people. They were happy to see the state's people being happy to be with it's people. They were happy to see the state's people being happy to be with it's people. They were happy to see the state's people being happy to be with it's people. They were happy to see the state's people being happy to be with it's people.

```
GPT-2 Salidas acumuladas: "The king has been very good at the military
and is now in charge. The majority of the population shared certain
believes. Vengrian Deities was their religion. Its form was Polytheism,
and focused on Balasztes, The Worried Father of Fire. The state of
Balombomia had Monarchy as a model of government."The king has been
very good at the military and is now in charge. The majority of the
population shared certain believes. Vengrian Deities was their religion.
Its form was Polytheism, and focused on Balasztes, The Worried Father of
Fire.  He's been with us for many years. You can't take him away from
you. This urban state was kind of outdated and couldn't move forward
without your help."  Arya looked up to see that Balombomian leader was
not just a man but an officer. She thought about how he'd come across
this position in time.  It was like being part of some great army.
He was also a member here in the city. He could have easily become
commander-in chief if he wanted to be. But he didn't want his people to
believe in him.
```

La segunda versión, a pesar de contar con muchas expectativas para ella, nos encontramos con varios problemas durante su implementación:

1. El aumento progresivo de la entrada del modelo, que incluye partes de la salida anterior, resultó en un tiempo de generación considerablemente más largo. Para que este enfoque funcionara adecuadamente, tuvimos que aumentar el parámetro *Max Length* del modelo para permitir salidas más largas. Esto afectó negativamente al rendimiento y el tiempo de ejecución del programa.

2. Observamos que a medida que incrementábamos la longitud de la entrada, el modelo generaba más errores. Se presentaron problemas como espacios en blanco excesivos o texto que no guardaba relación con el contexto previo. Esto afectó a la coherencia y la calidad de las salidas generadas.

3. En algunas ocasiones, el programa dejaba de ejecutarse, especialmente cuando se trabajaba con entradas más extensas. Experimentamos tiempos de ejecución de hasta 4 horas, lo cual resultaba ineficiente para las características de complejidad de nuestras entradas. Esto ocasionó interrupciones en el proceso y afectó a la productividad del proyecto.

En resumen, a pesar de que la inclusión de la salida previa en la entrada del modelo parecía prometedora para mejorar la coherencia, nos encontramos con desafíos significativos. Los problemas de tiempo de generación, errores en las salidas y la inestabilidad del programa nos llevaron a replantear el enfoque y buscar alternativas para garantizar resultados más eficientes y consistentes.

En el próximo capítulo, abordaremos el estado actual del proyecto, reflexionando sobre los avances realizados hasta ahora y discutiendo posibles mejoras, como los nuevos modelos sacados al mercado tales como GPT-4, y continuaciones para llevarlo a un nivel superior con las limitaciones del propio proyecto.

Capítulo 5

Discusión

A continuación vamos a discutir como se comparan los resultados obtenidos mediante nuestra herramienta comparados con otro tipo de soluciones, además de discutir las limitaciones presentes en la implementación actual.

5.1. Comparación con otras soluciones presentes en el mercado

En primer lugar, cabe comparar la solución obtenida con la ofrecida por otras aplicaciones similares presentadas en 2.4.

Por una parte discutamos las capacidades generalizadas presentadas por estas aplicaciones. En este ámbito podemos resumir una serie de características importantes:

- Utilización de modelos de generación de lenguaje natural avanzados como GPT-3, GPT-3.5... produciendo texto con menor repetición y más natural.
- Pago a través de suscripción para el uso de la mayor parte de las características. Al utilizar herramientas como GPT-3 la aplicación precisa de una remuneración acorde por parte del usuario, haciendo que la aplicación sea menos accesible para ciertos usuarios.
- Cierto grado de consistencia. Tanto NovelAI como AIDungeon cuentan con algún tipo de herramienta que permite introducir información que el modelo es capaz de recordar para salidas futuras. Esta memoria es limitada pero aporta resultados más coherentes con la historia.
- Precisan de una entrada de texto introducida por el usuario.
- En muchos casos la narrativa generada precisa de correcciones manuales por parte del usuario para solventar inconsistencias en la narrativa y/o errores.

Como podemos observar ninguna de las herramientas que ya estaban presentes en el mercado aportan una funcionalidad perfecta. El campo de la generación de

lenguaje natural se encuentra en continua expansión y evolución. Esta tecnología relativamente reciente sigue contando con una infinidad de errores, problemas de consistencia y errores sintácticos; incluso con modelos muy avanzados como GPT-3.5, la evolución de GPT-3.

Durante el desarrollo de nuestra herramienta tuvimos en cuenta dichas limitaciones y antes de continuar con el desarrollo introduciendo más elementos y complejidad en la generación, consideramos que era más importante y útil para el mercado ofrecer mejoras en todo este tipo de brechas. Nuestra aplicación busca ofrecer soluciones a todos estos problemas comunes:

- Problemas de sintáxis: eliminación de errores de sintaxis como símbolos erróneos producidos en la generación.
- Narración en tercera persona: nuestra herramienta al estar especializada en la generación de narrativa histórica cuenta con mecanismos para evitar la generación de texto con narración en primera persona.
- Sistema de puntuación: gracias a nuestro sistema de puntuación, nuestro sistema automáticamente elige la mejor salida obtenida. Permitiendo que el usuario no tenga que introducir la misma entrada varias veces hasta alcanzar una solución deseable.

Dichas mejoras son fácilmente implementables y funcionan de forma modular para cualquier modelo implementado en el futuro en nuestro programa, ofreciendo una gran capacidad de expansión y mejora. Consideramos este factor muy importante, debido a, como ya hemos mencionado anteriormente, la expansión constante del campo de la generación del lenguaje natural.

Por otra parte discutamos las características únicas de nuestra herramienta. En primer lugar y una de las principales ventajas de nuestra aplicación es la utilización de un mapa geográfico/político para la lectura y obtención de datos. Mientras que con otras implementaciones el usuario precisaría de la introducción manual de una descripción detallada de las características de cada una de las civilizaciones e ir entrelazando una historia, nuestra aplicación busca ofrecer toda esta funcionalidad de manera totalmente automática. Además, gracias a que el programa soportado para la generación del mapa cuenta con una cantidad enorme de características, el usuario siempre podrá introducir una entrada que le satisfaga en nuestro programa.

Más allá de esto, nuestra aplicación agrupa todas estas civilizaciones en un grafo, teniendo en cuenta todas las relaciones diplomáticas presentes entre ellos. Una vez mejorada la salida generada por los modelos hasta un punto deseable nuestra aplicación facilita enormemente la tarea de entrelazar todas estas salidas a generar, asegurando una correlación entre todas las historias.

5.2. Limitaciones de la solución propuesta

Tras analizar en detalle los resultados finales obtenidos por los modelos seleccionados en nuestro proyecto, hemos identificado tanto limitaciones como ventajas importantes, así como el estado actual en el que se encuentra el trabajo.

En cuanto a los generadores de texto utilizados, GPT-2 y GPT Neo, observamos que los resultados de los textos generados mejoraron considerablemente después de implementar el sistema de puntuación mencionado anteriormente. Sin embargo, como se puede ver en las salidas del modelo en el capítulo 4, encontramos varias limitaciones:

1. Una de las principales limitaciones de nuestro trabajo reside en la elección del modelo utilizado. En la actualidad, han surgido modelos más eficientes, como GPT-4, que ofrecen capacidades más amplias en comparación con su predecesor, GPT-2. Para mejorar nuestro enfoque, además de explorar nuevos modelos, podemos optimizar el proceso de re-entrenamiento del modelo de lenguaje pre-entrenado que utilizamos, como GPT-Neo. Al aumentar la cantidad de casos de entrenamiento, podemos esperar mejoras significativas en los resultados generados por el modelo. Sin embargo, es importante tener en cuenta que este proceso requiere una inversión adecuada de tiempo y recursos para llevarse a cabo de manera efectiva.

2. La cohesión entre los textos generados es un desafío incluso para los propios modelos de lenguaje, por lo que su mejora es una tarea pendiente que requerirá iteraciones futuras en el proyecto. Dado que nos enfrentamos a limitaciones de tiempo para su finalización en el presente trabajo, esta tarea de garantizar la cohesión se pospone para futuras mejoras del proyecto.

3. Por último, hay un aspecto pendiente que requiere una atención especial: dotar al texto generado de una estructura narrativa más completa. Hasta ahora, este aspecto ha pasado desapercibido debido a la necesidad de priorizar el progreso en el punto mencionado al inicio del capítulo. Actualmente, el programa selecciona un estado aleatorio de todos los existentes en el mapa y genera un texto que se centra únicamente en sus atributos y relaciones. Sin embargo, es menos relevante proporcionar una estructura a la conjunción de varios textos si aún no hemos asegurado que se relacionen entre sí. Por lo tanto, esta tarea de dotar al texto de una estructura narrativa coherente se considera pendiente y requerirá un enfoque más exhaustivo en futuras etapas del proyecto.

Hablando ahora de las fortalezas del proyecto, una de ellas radica en la estructura de nuestro código, que se centra principalmente en los textos generados por los modelos de lenguaje. Esta estructura nos proporciona flexibilidad para realizar cambios con facilidad, lo que nos permite probar modelos más recientes y avanzados que puedan surgir en el futuro con solo realizar pequeñas modificaciones en el código. Además, tenemos la oportunidad de reconsiderar los modelos que inicialmente descartamos debido a restricciones de pago, pero que podrían ofrecer mejoras significativas en los resultados.

Otra fortaleza destacada es la compatibilidad de nuestro proyecto con cualquier versión del mapa. El generador de mapas con el que trabajamos está diseñado para soportar diferentes versiones, lo que garantiza su utilidad a largo plazo. Además, hemos dejado el código preparado con la información completa del mapa y la estructura de grafos mencionada en el apartado 3.4, lo que facilita la continuidad del proyecto sin ningún problema.

En resumen, nuestro proyecto ha revelado limitaciones en términos de cohesión entre los textos generados y la falta de una estructura narrativa completa. Sin embargo, hemos trabajado en hacer que nuestro código sea lo más versátil posible para futuras versiones, permitiendo la incorporación de modelos más avanzados y la mejora del proceso de re-entrenamiento del modelo utilizado. Aunque el trabajo pendiente de proporcionar una estructura narrativa más completa ha sido relegado en pos de resolver las contradicciones, es un aspecto esencial que debe abordarse en futuras etapas del proyecto para lograr resultados más satisfactorios.

En el último capítulo presentaremos las conclusiones finales al comparar los resultados obtenidos con los objetivos iniciales, y exploraremos posibles expansiones futuras.

Capítulo 6

Conclusiones y Trabajo Futuro

Para finalizar, vamos a revisar las conclusiones obtenidas con respecto a los objetivos establecidos inicialmente.

6.1. Generación de mapas

En primer lugar, en cuanto a la generación del mapa, logramos abordar este objetivo mediante el uso de una herramientas de terceros, permitiendo que nuestro programa sea compatible con dicho mapa. Esto significa que hemos logrado que el programa pueda leer un mapa con la información necesaria para la generación narrativa. Además, cabe destacar que nuestro programa es capaz de funcionar con cualquier actualización futura del generador de mapas, lo cual garantiza su continuidad en el futuro sin problemas.

6.2. Generación de texto

En segundo lugar, uno de los mayores desafíos de este proyecto ha sido la búsqueda de un modelo que se ajuste a nuestras necesidades. La utilización de GPT-2 y GPT-Neo ha aportado un prototipo inicial fácilmente ampliable en el futuro debido a la modularidad de nuestro trabajo. Una gran cantidad de modelos están surgiendo día tras día y nuestra aplicación será capaz de implementarlos con mínimas modificaciones en el código.

Fine-tuning

Nuestros intentos de realizar un fine-tune del modelo GPT-Neo, no arrojaron mejores resultados que los producidos por el modelo del que ya disponiamos, GPT-2. Esto es debido principalmente a la falta de casos de entrenamiento y la carencia de una base de datos suficientemente extensa y adecuada para el proyecto. El proyecto aún así aporta una base sobre la que realizar dicho entrenamiento, el cuál puede ser realizado fácilmente dada una base de datos apropiada.

6.3. Sistema de puntuación

Asimismo, hemos intentado abordar este problema mediante el desarrollo de herramientas para evaluar las frases generadas y guiar mejor a los modelos, cuyos resultados fueron positivos, evitando *Word Looping* y *Looping Behaviour*, al mismo tiempo que reduciendo la aparición de dialogos, y seleccionando la mejor salida entre varias generadas. Estas herramientas marcan una mejora sustancial en la resolución de dichos problemas, generando resultados mucho más naturales.

Además, hemos logrado crear un sistema capaz de evaluar la salida del modelo y seleccionar la mejor opción entre todas las generadas. Esta herramienta puede ahorrar mucho tiempo al usuario, generando respuestas de mayor calidad de forma totalmente automatizada.

6.4. Cohesión de textos

Por último lugar, la generación de un documento de texto que consta de historias coherentes e interconectadas: observamos que tanto GPT2 como GPT Neo, dos modelos basados en transformers, no lograron mantener la consistencia necesaria por sí solos en la generación de textos. Además, nos encontramos con dificultades para que el sistema de puntuación pudiera evaluar de manera precisa la cohesión de los textos generados por estos modelos. Dada la complejidad de la tarea y el tiempo limitado con el que contábamos, decidimos ajustar nuestro enfoque y concentrarnos en la generación de texto para una sola parte del mapa. Esta decisión se basó en la necesidad de obtener resultados concretos y abordables dentro de nuestras limitaciones de tiempo y recursos. Sin embargo, la implementación de un grafo con relaciones entre los distintos estados permite una gran base para unir dichos textos, la cuál será muy útil en el desarrollo futuro del proyecto.

En conclusión, hemos logrado cumplir con el objetivo de desarrollar un programa capaz de leer y utilizar la información de un mapa para la generación narrativa. No hemos podido disponer de un modelo que se ajuste completamente a nuestras necesidades para mantener la coherencia y calidad necesarias en la generación de textos, motivo por el que desarrollamos diferentes herramientas que mejoran cualquier salida obtenida independientemente del modelo. Algunos mecanismos como el sistema de puntuación, corrección de sintaxis y el *fine-tuning* ofrecen una gran cantidad de herramientas muy útiles en la generación de narrativa. Consideramos que dichas herramientas mejoran sustancialmente el resultado obtenible y marcan un gran paso en la realización del proyecto.

6.5. Trabajo futuro

Como posibles formas de continuar con el trabajo hemos identificado varias sugerencias y posibles direcciones a seguir:

- Emplear un modelo que pueda mantener una mínima consistencia en la gene-

ración de textos narrativos. Dado que nuestro código está estructurado para facilitar esta implementación, sería relativamente sencillo incorporar un modelo más consistente. Además, el sistema de corrección de frases que ya hemos desarrollado podrían mejorar aún más las respuestas generadas por este modelo. Aunque modelos como ChatGPT tienen limitaciones debido a su alta demanda,restricciones de acceso y recursos financieros, explorar su uso o investigar modelos similares podría ser una opción. En caso de que estos modelos no logren mantener la consistencia requerida en un mapa extenso, sería necesario desarrollar un sistema de consistencia específico.

- Elaborar una base de datos de historias similares a las que se buscan en este proyecto. Hasta la fecha de este TFG, no hemos encontrado una base de datos adecuada para entrenar un modelo que genere historias a partir de un texto en formato de narrador. La creación de una base de datos extensa y apropiada sería un esfuerzo considerable, pero una vez completada, mejoraría significativamente las respuestas proporcionadas por los modelos. Esto requeriría recopilar historias relevantes y estructurarlas en un formato adecuado para su uso en el entrenamiento del modelo.

- Relacionar las diferentes partes del mapa para obtener historias interconectadas que sean útiles para el usuario. Actualmente, hemos centrado nuestros esfuerzos en la generación de texto para una parte menor del mapa. Sin embargo, sería valioso expandir esto y desarrollar un sistema que pueda relacionar de manera coherente y significativa las distintas partes del mapa para generar historias interconectadas. Esto permitiría a los usuarios explorar y disfrutar de narrativas más completas y cohesionadas en el contexto del mapa.

Chapter 7

Introduction

This last year, AIs have been very frequently the main topic of conversation between computer experts. AI is a field with constant growth and there has already been an upsurge of very advanced models. From models like Dall-e, an AI system that can create realistic images and art using just a written description in natural language (OpenAI (2021)). Or Midjourney's models, an independent research team centered around the design, the human infrastructure and Artificial Intelligence, that has developed various different models that revolve around image generation (Midjourney (2022)). On the other hand, there also exist AIs like ChatGPT, a trained model that interacts from a conversational standpoint and is capable of answering different questions, admit it's mistakes, and challenging incorrect premises (OpenAI (2022)).

There are various models of narrative text generation focused on the creation of novels, tales, and similar works. However, we have observed the lack of a model exclusively focused on the generation of narrative worlds. That is why we wonder if it would be possible to use one of the existing models to create a story that includes information about civilizations, religions, cultures, conflicts, and more, starting from a map. Consequently, we are drawn to the idea of creating a system capable of generating in-depth stories about civilizations, based on a detailed map that provides socio-political information, using existing narrative generation models. Thus, the basic idea of our project emerges: the development of a tool that takes certain customizable parameters from the user and creates a story guided by the user that is sufficiently complete to be used or serves as creative inspiration for the development of other projects.

7.1. Motivation

This projects is intended to offer story generation upon a world to anyone that needs it. A lot of people, for differente reasons, could need from the creation of a narrative world, for example, a novel writer could need a world in which to write a story, or an artist could be looking for inspiration for a painting, or a game master could need a world in which to base a game; and our tool pretends to help in all those situations, inspire and offer a creative base to anyone who needs it.

Created stories don't have the intention of becoming works, or substituting the work of a human, but to offer information in a superficial way about a series of civilizations, cultures and religions to make up the base of an even bigger project. As it doesn't specify in depth any occurred event, or talk about specific characters, a basic socio-politic structure is offered which the user can give a more specific use depending on their needs.

7.2. Objectives

The project's objective consists in the development of a tool capable of generating interconnected stories from a geographic/political map, with different civilizations, cultures, religions... The initial map must be able of being configured by the user prior to it's utilization in our program, allowing the most possible degree of creative freedom. The resulting program will respect certain parameters introduced by the user, generating a series of stories that fit with those generated previously. The result of this generations will be shown to the user in plain text.

The story generation will make use of different free language models that allow the use of the program to any kind of user. Moreover, different sintax correction and narrative consistency tools will be implemented in order to offer better results with any utilized of model.

7.3. Methodologies

In the development of our project, we have applied agile methodologies, specifically Scrum, to ensure an efficient and flexible approach in the planning and execution of tasks. Initially, during the research phase, considering our limitations due to academic commitments, we set research goals for periods of two to three weeks, focusing on exploring and understanding the most relevant concepts for the project.

Starting in January, when we focused on the development of the tool, we adjusted the timelines and adapted our methodology to weekly sprints, known as "weeklys," with the aim of progressing more quickly and effectively. During these sprints, we assigned specific tasks to each team member and regularly met to review the code and individual progress, which allowed us to stay updated on the project as a whole and understand its integral functioning.

Additionally, we made an initial estimation of the work that reflects the distribution of tasks over time, but we adapted flexibly as we progressed, following Scrum principles. Throughout the project development, we constantly evaluated changing needs and priorities, adjusting the distribution of tasks over time. Taking into account our academic commitments and other limitations, we set realistic and achievable goals at each stage of the project. This has allowed us to maintain a steady pace and ensure continuous progress towards our objectives.

For the creation and organization of project tasks, as well as their assignment, we decided to use Discord. Discord is a free remote communication and socialization program that incorporates text, voice, and video chats. In this program, users can

create servers and invite others to participate in the voice or text channels it offers (Schwartz, 2021). We created a server for the development of the work, enabling us to participate in different channels we created to separate information, such as a bibliography channel, progress made, errors in the report, etc. This way, we well-structured our goals and research and avoided losing links to investigations or work.

We have estimated the work planning in table 7.1, taking into account our limitations with project progress due to academic matters.

In summary, we have applied agile methodologies, particularly Scrum, in our development approach. We have carefully planned and adapted our methodology as we progressed in the project, allowing us to optimize our time and resources, as well as stay aligned and committed to our objectives.

Figure 7.1: Planification of pending work from the beginning of the academic year until the end date

7.4. Work plan

In order to fulfil the previously mentioned objectives, we divided our work to begin development in the following manner:

1. Look for a simple way of generating a map, with custom features and, at the same time, find a way of turning them into parameters, by a generator or a tool capable of reading it, and allowing us to export the data in a format legible by a human, and that's useful for our project.

2. Researching the different Natural Language Generation Models available publicly at the moment, that could be ised for story generation starting from a

data input, and selecting the ones that could offer the most benefits to our project.

3. Reading the generated map, storing the data in a structure, selecting the ones that are most relevant for the creation of the story, and transforming them into an input that can be processed by the model.

4. Loading the previously selected models, creating prototypes for a starting data input that's simple, making a comparison of the outputs provided by each one, and choosing the model with the best results.

5. Improving the quality of the obtained outputs, making them more developed and complex, contributing a wider variety of inputs top avoid repetition. As well as, increasing the quantity of map characteristics that are used and modifying the model parameters to increase the length of the generated text.

6. Create an evaluation system that, given a set of outputs, selects the best formulated one to avoid possible errors and increase text consistency.

7. Iterate between the two previous steps to refine the project and improving obtained results, and selecting a group of people that can offer feedback.

8. Generate a text document with the stories generated by our tool.

7.5. Document structure

This document is divided into different sections which will be explained below:

1. As previously shown, in chapter 1 we make a summary of the project, mentioning the purpose, our goals with the creation of the tool and the context on which it can be used.

2. In chapter 2 we indicate the prior investigation about map generators, the different language models for the processing of text, the investigation of a machine learning technique to adapt one of our selected language models called *Fine-tuning* and an investigation regarding other alternative story generators currently present in the market.

3. In chapter 3 we talk about the development of the tool and how did we have to split the work to progress simultaneously. On the one hand, testing and selecting the language models mentioned previously in chapter 2 and comparing the outputs to choose the best result. On the other hand, taking into account the limitations of the map generatirs and the way of utilizing all of their information and storing it in a graph in the project. And, lastly, the implementation of a point system that evaluates the models outputs in order to generate a better result. Betterment in the models outputs to increase the amount of text that it generated is also mentioned, as well as the implementation of the *Fine-tuning* mentioned previously in chapter 2 for each of the selected models.

4. In chapter 4 we talk about the results obtained with our tool.

5. In chapter 5 we discuss the limitations found in our project, as well as it's possible advantages.

6. Lastly, in chapter 6, we give a conclusion to this project. We cover the accomplished goals, as well as the limitations found along the way, the problems we have encountered, how to improve the program in the future, and possible future extensions of the project.

Conclusions and Future Work

To conclude, we are going to review the conclusions obtain regarding the initially established objectives.

7.6. Map generation

Firtsly, regarding the map generation, we succesfully addressed this objective by using third-party tools, enabling our program to be compatible with said map. This means we have ensured that the program is able to read a map with the necessary information for narrative generation. Additionally, it is worth mentioning that our program is capable of functuioning with any future updates to the map generator, guaranteeing its continuity in the future without any issues.

7.7. Text generation

Secondly, one of the major challenges of this project has been finding a model that fits our needs. GPT-2 and GPT-Neo have provided an initial prototype easily extendable in the future due to the modularity of our work. A great number of models are making their appearance day by day and our program will be capable of implementing them with minimum changes in the code.

7.7.1. Fine-tuning

Our attempts to fine-tune the GPT-Neo model did not yield better results than the model we already had, GPT-2. This is primarily due to the lack of training data and the absence of a sufficiently extensive and suitable database for the project. The project, however, supplies a base with which to perform said training, which can be easily performed with an appropriate data base.

7.8. Scoring system

Furthermore, we have attempted to address this problem by developing tools to evaluate the generated sentences and better guide the models. The results were

positive in terms of avoiding Word Looping and Looping Behavior, reducing the occurrence of dialogues, and selecting the best output among several generated ones. These tools mark a substantial step forward in solving these problems, generating way more natural results.

Moreover, we have accomplished the creation of a system capable of evaluating the model's output and selecting the best option from a series of generations. This tool can save a lot of time for the user, generating better answers automatically.

7.9. Text cohesion

Lastly, the generation of a text document consisting of coherent and interconnected stories: we observed that both GPT-2 and GPT-Neo, two transformer-based models, were unable to maintain the necessary consistency on their own in text generation. Additionally, we encountered difficulties in accurately evaluating the cohesion of texts generated by these models using the scoring system. Given the complexity of the task and the limited time available to us, we decided to adjust our approach and focus on text generation for a single part of the map. This decision was based on the need to obtain concrete and manageable results within our time and resource constraints. However, the implementation of a graph with connected nations offers a great starting point to join all this stories, which will be very useful in the future development of the project.

In conclusion, we have successfully achieved the objective of developing a program capable of reading and utilizing map information for narrative generation. We didn't have a model at our disposal that could meet our needs of maintaining the much needed consistency and quality for text generation, reason why we developed different tools that improve the quality of any output, independently from the model. Some mechanisms like the score system, syntax correction and the *fine-tuning* offer a great amount of tools useful for narrative generation. We consider that said tools improve the obtained result substantially and mark a big step in the development of the project.

7.10. Future work

As possible ways to continue the work, we have identified several suggestions and possible directions to follow:

- Employ a model that can maintain a minimum level of consistency in narrative text generation. Since our code is structured to facilitate this implementation, it would be relatively straightforward to incorporate a more consistent model. Additionally, the sentence correction system we have already developed could further improve the responses generated by this model. Although models like ChatGPT have limitations due to high demand, access restrictions, and financial resources, exploring their use or researching similar models could be

an option. If these models fail to maintain the required consistency in an extensive map, it would be necessary to develop a specific consistency system.

- Create a database of stories similar to those sought in this project. Up until the date of this bachelor's thesis, we have not found a suitable database to train a model that generates stories from a narrator's text format. Creating an extensive and appropriate database would be a considerable effort, but once completed, it would significantly enhance the responses provided by the models. This would involve collecting relevant stories and structuring them in a suitable format for use in model training.

- Establish connections between different parts of the map to obtain interconnected stories that are useful for the user. Currently, we have focused our efforts on text generation for a smaller portion of the map. However, it would be valuable to expand this and develop a system that can coherently and meaningfully relate the various parts of the map to generate interconnected stories. This would enable users to explore and enjoy more comprehensive and cohesive narratives within the context of the map.

Contribuciones Personales

Estudiante 1: Sam Blázquez Martín

Inicialmente no teníamos una idea sólida, por lo que realizamos una investigación exhaustiva para seleccionar entre dos ideas iniciales: un generador de historias narrativas o un simulador de civilizaciones. Investigé simuladores populares en el mercado y revisé Trabajos de Fin de Grado previos relacionados con ambas ideas para evaluar su alcance y viabilidad.

Una vez decididos por el generador narrativo, me sumergí en una investigación detallada sobre los modelos de lenguaje que podríamos utilizar. Investigué sobre aprendizaje automático, deep learning y modelos Transformers, como BERT, Beto, MarIA, GPT-3, GPT-Neo y GPT-2, pues nunca habíamos trabajado con ningún modelo de lenguaje. Analicé sus características, ventajas, desventajas y modelos pre-entrenados para implementarlos en nuestro proyecto y realizar pruebas.

Participé en la investigación de los generadores de mapas más populares del mercado y evalué sus ventajas y desventajas. Trabajé con Azgaar's Fantasy Map Generator y me familiaricé con su interfaz y funcionalidades. Establecí contacto con moderadores de su servidor de Discord para obtener información adicional sobre el uso de ese mapa específico y recibir orientación en su funcionamiento.

Investigué la integración del modelo GPT-3 en nuestro proyecto y exploré sus capacidades. Descubrí la funcionalidad Ïnserting Text"que se encontraba en fase beta, que permitía completar el texto de entrada con una salida relacionada. Sin embargo, debido a su alta demanda y métodos de pago basados en *tokens*, decidimos descartar esta opción y continuar investigando modelos gratuitos como GPT-2 y GPT-Neo.

En colaboración con Alberto Muñoz, busqué e implementé modelos pre-entrenados de Hugging Face, además de familiarizarme con la página (pues tampoco habíamos trabajado con ella) e investigar acerca de su api y el funcionamiento de su apartado de Modelos. Estuvimos seleccionando los modelos que más se aproximaban a lo que buscábamos,metiéndoles en el prototipo inicial, evaluando sus salidas y seleccionando aquel que mejor se adaptara a nuestros objetivos. También exploré las funcionalidades de ChatGPT y su personalización, aunque finalmente fue descartado debido a su alta demanda y sistema de pago.

Me enfoqué en pulir y mejorar la estructura del proyecto, desarrollando un se-

gundo prototipo con una estructura de grafos para extraer y almacenar de manera correcta los datos relevantes del mapa en formato JSON.

Junto con Alberto Muñoz, diseñé un sistema de puntuación para evaluar la calidad de las salidas generadas. Investigamos y utilizamos diversas librerías de Python, como nltk, re, PyTorch, difflib y Spacy. Contribuí a la detección y puntuación negativa de eventos y lugares famosos, errores de sintaxis y diálogos en primera persona. También trabajé en un criterio para evaluar la coherencia de eventos entre dos textos, aunque esta funcionalidad no se implementó en la versión final debido a la necesidad de refinar el método y limitaciones de tiempo.

Posteriormente, cuando nos decantamos por el generador narrativo, comencé investigando en profundidad sobre los modelos de lenguaje que podíamos utilizar. Para entender los modelos de lenguaje hubo una investigación previa acerca del aprendizaje automático y el *deep learning* para entender así el funcionamiento de los modelos. Durante el proceso descubrí los llamados Transformers y sus diferentes tipos mencionados en el Estado de la Cuestión, como BERT, Beto, MarIA, GPT-3, GPT-Neo y GPT-2. Analicé sus ventajas, desventajas, repositorios y modelos pre-entrenados para implementarlos y realizar pruebas. También me sumergí en la comprensión del funcionamiento de un Transformer, ya que ninguno de los tres miembros del equipo había trabajado previamente con modelos de lenguaje.

Además, participé en la investigación de los generadores de mapas más populares del mercado. Me informé acerca de sus pros y cons y posteriormente trabajé con cada uno de ellos para seleccionar el más adecuado para nuestra tarea. Al seleccionar Azgaar's Fantasy Map Generator me sumergí en el proceso de entendimiento de su interfaz, así como maneras de explotar al máximo la herramienta. Estuve en contacto con moderadores de su servidor de Discord para informarme acerca del uso que podíamos tener con ese propio mapa, además de que me facilitó ayuda para entender el funcionamiento de la herramienta.

Cuando GPT-3 fue lanzado, investigué cómo integrarlo en nuestro proyecto y cómo podríamos aprovechar su potencial. Descubrí una funcionalidad en fase beta llamada Ïnserting Text"que se ajustaba a nuestras necesidades, ya que permitía completar el texto de entrada con una salida relacionada. Sin embargo, debido a su alta demanda y métodos de pago basados en *tokens*, decidimos descartar esta opción y seguir investigando modelos gratuitos. Después de descartar GPT-3, nos centramos en GPT-2 y GPT-Neo. Continué probando ambos modelos para analizar cuál sería más adecuado para nuestro proyecto. Participé en el desarrollo de un prototipo inicial donde comparamos las frases generadas por ambos modelos y exploramos cómo personalizar la salida modificando sus parámetros para aprovechar al máximo sus capacidades.

En colaboración con Alberto Muñoz, me involucré en la búsqueda e implementación de modelos pre-entrenados de *Hugging Face*. Evaluamos sus salidas y determinamos cuál se acercaba más a nuestra idea. Luego, cuando ChatGPT fue lanzado, exploré sus funcionalidades y hasta dónde podíamos personalizarlo. Sin embargo, al igual que GPT-3, fue descartado debido a su alta demanda y sistema de pago.

Una vez que tuvimos la estructura definida, me enfoqué en pulirla y mejorarla. Desarrollé el segundo prototipo con la estructura del grafo para extraer y almacenar correctamente los datos más relevantes del mapa en un archivo JSON que utiliza-

ríamos en el modelo. En este segundo prototipo, creé junto con Alberto Muñoz el sistema de puntuación que evaluaba la calidad de las salidas generadas. Investigamos y utilizamos diversas librerías de Python, como nltk, re, PyTorch, difflib y Spacy.En mi caso, para la creación del sistema contribuí a la detección y evaluación de eventos y lugares famosos, errores de sintaxis y diálogos en primera persona. También trabajé en un criterio para evaluar la coherencia de eventos entre dos textos, aunque esta funcionalidad no se implementó en la versión final debido a la necesidad de refinar el método y limitaciones de tiempo.Sin embargo, el código se mantuvo en el proyecto por una posible expansión de futuro.

Todos los miembros del equipo colaboramos en el último prototipo donde nos centramos en la mejora de la salida generada y la limpieza del código. Aseguramos que el entorno de Google Colab estuviera adecuadamente comentado para facilitar la comprensión a personas externas al proyecto. Realizamos pruebas exhaustivas de los modelos y evaluamos sus resultados en relación con los objetivos iniciales. Además, comparamos los resultados obtenidos y reflexionamos sobre posibles mejoras y soluciones a futuro para continuar perfeccionando el sistema

Estudiante 2: Miguel Hernández García

Yo estuve estudiando la viabilidad de los diferentes recursos de generación de mapas disponibles. Al ser un proyecto muy centrado en las civilizaciones y como estas se comportarían en correspondencia con el terreno, otras civilizaciones y demás factores, buscábamos un generador de mapas que le permitiera al usuario introducir y jugar con todos estos valores.

Un recurso que encontré después de un tiempo extenso de investigación fue un mod del juego Civilization que permitía que la inteligencia artificial jugara partidas por si sola sin intervención del usuario. Esto nos permitía realizar partidas automáticas que nos proporcionaran con un mapa del que partir, además de unos confrontamientos que luego narrar y generar historias a partir de ellos. Además, la gran complejidad y extensión de los mapas de Civilization nos aportaría con todos aquellos elementos que contaban con un gran potencial narrativo, pero que serían muy costosos como para desarrollarlos nosotros de cero. Este mod fue probado en Civilization y dio un resultado satisfactorio, una partida automáticamente generada que "simulaba.[el] paso del tiempo para esas civilizaciones. Estos datos eran almacenados en unos logs para su posterior procesamiento. Los logs en cuestión estaban divididos en distintos ficheros y contenían mucha información innecesaria para nosotros, por lo que tomaría una cantidad sustancial de tiempo realizar una herramienta que recogiera esos datos y los convirtiera en una estructura con la que pudiéramos trabajar externamente.

Esta alternativa nos permitía utilizar un mismo recurso tanto para la generación del mapa como para una simulación del paso del tiempo que nos aportara material que narrar. Fue discutido con el resto de integrantes del grupo y con los profesores supervisores y al final fue descartada debido a que la idea se parecía mucho a un TFG pasado, por lo que decidimos probar con otras alternativas.

Después de decidirnos por el generador de mapas final, me encargué de leer e

interpretar los datos del archivo .json que representaba el mapa para que el resto de integrantes del grupo encontraran más fácil su utilización y facilitar lo más posible la lectura de los distintos datos. Realicé distintos bucles que recorrían el diccionario obtenido del mapa que simulaban su recorrido y luego escribiéndolo en la salida. Además de incluir distintos comentarios descriptivos y señalar aquellas variables que encontraba personalmente más útiles a la hora de narrar una historia.

También implementé un modelo de GPTNeo como alternativa al modelo de GPT2 implementado previamente, así como unos prompts básicos para probar ambos modelos. Estos prompts fueron los empleados por Sam Blázquez para la evaluación preliminar de ambos modelos y decidir con que modelo quedarnos para el resto de la investigación.

En seguida nos dimos cuenta de las limitaciones de ambos modelos. Tanto GPT2 como GPTNeo narraban de una manera no muy convincente, prefiriendo muchas veces las narraciones en primera persona, o de personajes en concreto. En vez de pensar en una escala algo más global y a escala de civilización como tal. Además, no mantenían ningún tipo de coherencia claramente entre los outputs y ni siquiera en un único output si este era lo suficientemente largo. Empezaban a repetir frases continuamente en algunos casos y muchos otros problemas. Decidimos probar distintas maneras de solucionar estos problemas aún sabiendo que nos tomaría bastante tiempo. A nuestros ojos, cuanto más nos acercáramos a un output más satisfactorio, luego todas esas herramientas realizadas se podrían utilizar como base para darle toda la complejidad que quisiéramos al resto de la estructura del programa.

Por mi parte intenté arreglar los problemas presentes en GPTNeo utilizando fine-tuning, o dicho de otra manera, sobreentrenando el modelo utilizado anteriormente de GPTNeo con más inputs y outputs que se parecieran más a los deseados. En este recorrido, el principal problema que me encontré fue conseguir una base de datos lo suficientemente grande y que tuviera unos datos que fueran compatibles con nuestro modelo. La gran mayoría de bases de datos disponibles por internet son bases de datos de clasificación, en el que mediante un texto obtienes un resultado numérico o de otro tipo que clasifica o identifica el texto. También hay muchas bases de datos de procesamiento numérico. En concreto, estaba buscando una base de datos que narrara conflictos o acontecimientos entre civilizaciones/naciones. Encontré algunas páginas que compilaban acontecimientos históricos, entre ellas una página de George Mason University fue bastante útil al contener un glosario de distintas bases de datos con conflictos y demás acontecimientos. Sin embargo, muchas de estas referencias ya no son accesibles y las que sí que son accesibles, como por ejemplo: [1] eran dificilmente exportadas a un archivo utilizable para fine-tuning. Esta última en concreto es bastante interesante porque contiene cientos de artículos fruto de una investigación de 25 años por Michael Brecher, Jonathan Wikenfeld, Patrick James, Kyle Beardsley y David Quinn. Estos datos fueron anteriormente publicados como parte del libro A Study of Crisis, por Michael Brecher y Jonathan Wikenfeld y la Prensa de la Universidad de Michigan. Sin embargo, la exportación de estos datos tomaría una gran cantidad de tiempo y trabajo manual. En cambio, en el resto de bases de datos, puedo obtener acceso de manera muy sencilla a una serie de datos numéricos, pero nuestro interés no recae en esos datos sino en la redacción de los propios conflictos,

[1] `http://www.icb.umd.edu/dataviewer/`

volviendo estos datos inservibles para lo que buscaba. Desafortunadamente, para realizar el fine-tuning necesitaba obtener de cualquier manera una base de datos. Por ello, decidí aportar las horas necesarias y exporté la gran mayoría de los conflictos narrados en la web http://www.onwar.com/index.html y reunirlos en un archivo llamado conflictData.json en un formato que permitiera utilizarlo como base de datos para un fine-tuning. Esta página contiene todos los conflictos ocurridos entre los años 1800 y 1999, ambos incluidos. Fue un proceso largo pero que nos proporcionó una base de datos manual de cerca de 1000 conflictos, estas narraciones se acercan más a lo que buscamos y aunque contienen algunos comportamientos como inclusión de eventos reales, personas reales y demás que no nos convencían como grupo, no teníamos otra opción ya que redactar de nuevo estos acontecimientos nos costaría una cantidad de tiempo con la que no contábamos.

Posteriormente, una vez realizada la base de datos procedí a intentar realizar el fine-tuning, nunca había realizado algo parecido, ya que en nuestra carrera nos enseñan a entrenar redes neuronales, pero nunca había llegado a trabajar con un trainer o a tokenizar datos. Pero gracias a diversas fuentes y mucho ensayo y error conseguí hacer el fine-tuning del modelo y exportar el resultado a Google Drive para su rápida reutilización. Este modelo fue compartido con el resto de integrantes del grupo. Además, investigué como montar la unidad de Google Drive en la sesión de Google Colab y a partir de esa unidad montada acceder a los archivos necesarios, que se encontraban en rutas distintas dependiendo de si era un fichero compartido, o, en mi caso, se encontraba en mi propia unidad. Posteriormente realicé una generación de ejemplo e investigué como utilizar el nuevo modelo importado, ya que a diferencia de los modelos importados de Hugging Face, este modelo requería manualmente convertir el input en unos datos legibles por el propio modelo (encoding) y posteriormente convertir el resultado en algo legible por nosotros (decoding). Facilité un ejemplo de uso para facilitar que el resto del grupo solo tuviera que sustituir la generación normal por ese fragmento de código y así poder probar el modelo con el resto de herramientas.

El resultado no fue el esperado y el modelo resultó tener aún bastantes problemas, probablemente por el escaso número de casos nuevos en la base de datos que realicé. Probablemente se obtuviera un resultado más satisfactorio desarrollando una base de datos con un número elevado de casos y con tiempo para pulir dichos casos, pero esa cantidad de trabajo probablemente fuera prácticamente suficiente para otro TFG entero. Por lo tanto, decidimos simplemente dejar las herramientas necesarias para el entrenamiento, en caso de que alguien quiera retomar donde lo dejé. Probablemente, ayude también añadir un mayor número de iteraciones (epochs) en el fine-tuning, pero en nuestro caso, con la base de datos con la que contábamos el resultado no fue lo suficientemente diferente.

Estudiante 3: Alberto Muñoz Fernández

Mis contribuciones comenzaron con la búsqueda de posibles generadores de mapas de los cuales pudiéramos extraer los datos necesarios para basar nuestra historia. Como se mencionó en el capítulo del Estado de la Cuestión 2, buscábamos un mapa

que generara un contexto geográfico sobre la situación de varias civilizaciones en él, y al mismo tiempo proporcionara pistas para generar un contexto socio-político sobre las relaciones entre ellas. Inicialmente, además de los generadores de mapas, investigamos simulaciones que ejecutaran estos mapas. Esto me llevó a investigar simuladores de mapas simples como Civilization Simulator 2 [2], que encontramos en un repositorio de GitHub. Este fue el ejemplo que estuvo más cerca de ser integrado en nuestro proyecto, pero investigamos muchos más. La mayoría consistía en el desarrollo de civilizaciones basado en el azar, es decir, los conflictos y su desarrollo se basaban en la aleatoriedad.

Más adelante, decidimos descartar los simuladores, ya que nuestro trabajo se limitaría a narrar lo que ocurre en un programa que no creamos nosotros, y además no era el resultado que realmente buscábamos. Fue entonces cuando comencé a investigar los generadores de mapas descritos en el estado de la cuestión, y también realicé pruebas para determinar su efectividad y las ventajas que tendrían en nuestro trabajo. Esto incluía asegurarme de que se pudiera extraer la información del mapa en un formato legible por código, que su generación fuera sencilla y rápida, y que permitiera cierta personalización por parte del usuario.

Así fue como encontramos Azgaar's Fantasy Map Generator 2.1.4, del cual mi compañero Miguel se encargó de la lectura de los datos en formato JSON para nuestro proyecto. Recopilé los datos más relevantes y los utilicé para el desarrollo del primer prototipo, como mencionaré más adelante.

Antes de hablar del primer prototipo, mientras investigaba sobre los transformers, ayudé a Sam en la búsqueda de información sobre modelos de generación de lenguaje. Además, cuando la fase de investigación pasó a la prueba e incorporación de estos modelos, busqué e implementé en su mayoría los modelos de GPT-2 y GPT-Neo pre-entrenados que se encontraban en*Huggin Face*. Después de examinar a fondo los resultados proporcionados por los modelos, seleccionamos dos y comencé a desarrollar los primeros prototipos del proyecto.

Volviendo al desarrollo del primer prototipo, este incluía, como mencioné antes, el análisis y extracción de los datos más importantes que proporcionaba el JSON exportado del generador de mapas, y el uso de los modelos de lenguaje GPT-Neo y GPT-2 para generar salidas basadas en esos datos. Como se mencionó en el apartado correspondiente 3.3, fue un prototipo sencillo que solo verificaba un parámetro que determinaba la expansión de dos estados seleccionados al azar, y en función de esto, se proporcionaba una entrada al modelo que dictaba si entrarían en guerra o estarían en paz.

Durante el desarrollo de los prototipos y con el lanzamiento de ChatGPT, también investigué una API de Python que permitía su uso desde el código. La implementación de esta API no era compatible con Google Colab, ya que en realidad implicaba conectarse a la web y hacer consultas en tiempo real a ChatGPT, y como Colab se ejecuta en una máquina virtual, impedía la ejecución de métodos asíncronos en los que se basaba la API. Por un breve momento, propuse a mis compañeros cambiar al entorno de trabajo de Jupyter, donde sí podríamos ejecutar estos métodos asíncronos, con la esperanza de utilizar ChatGPT como generador de texto en

[2]cita https://github.com/ben9583/civilization-simulator

nuestro proyecto. Sin embargo, abandonamos esta investigación debido a la incertidumbre sobre la disponibilidad de ChatGPT en el futuro y a la frecuente bloqueo de la página debido a la alta demanda.

Después, cuando ya teníamos una estructura definida en el proyecto, me enfoqué en el desarrollo de un sistema de puntuación que evaluara los textos generados por los modelos y seleccionara el mejor resultado en función de su calidad. Específicamente, desarrollé un sistema que detectaba repeticiones de palabras en una misma frase, detecciones de frases repetidas o similares, y la detección de diálogos en el texto. Para ello, tuve que investigar diferentes bibliotecas de Python, como difflib, que se utiliza para comparar secuencias de datos, incluyendo cadenas de texto generadas por nuestros modelos, o nltk, que tokeniza el texto para crear una lista de palabras que utilizamos para la detección de repeticiones. Para evitar la aparición de diálogos, también tuve que investigar la biblioteca re, que casualmente Sam también investigó para evitar la aparición de eventos reales en el texto generado. En ocasiones, por ejemplo, se mencionaba "Nueva York". Estos sistemas de evaluación los reuní en un método que consistía en un bucle que se ejecutaba hasta 10 veces. En cada iteración, en función de los sistemas de evaluación, se les asignaba un valor que determinaba su calidad, y se guardaba solo el texto con el valor más bajo, que sería el generado finalmente. En futuros prototipos, cambié este método para que, en caso de aparición de diálogos o eventos reales, el texto se generara de nuevo y ni siquiera tuviera la opción de comparar su puntuación con otros textos.

Después de completar la implementación de esto, complejizamos el prototipo para ponerlo a prueba, añadiendo más información obtenida del mapa. Sam desarrolló un sistema de grafos que almacenaba la información que inicialmente determiné como relevante, y adapté el prototipo a los grafos. Además, aprovechando la estructura de los grafos, incrementé los datos utilizados en la generación, incluyendo, por ejemplo, su religión, su tipo de gobierno, si era o no expansionista (como inicialmente), si era urbano o más avanzado, y si la civilización estaba en avance o en declive.

Finalmente, diferencié dos tipos de algoritmos: los que recogían el prompt anterior para generar el siguiente, y los que generaban los prompts de manera individual y luego los unificaban. Explicándolo de manera más clara, como se desarrolló a lo largo de la memoria, los modelos necesitaban una entrada de texto para generar el texto de salida. Con el objetivo de aumentar la consistencia entre los textos generados, utilicé los textos generados previamente como entrada para la generación del siguiente texto. Por otro lado, en otro prototipo simplemente se generaban los textos y luego se concatenaban al final. Comparando sus resultados e investigando cuál de ellos proporcionaba mejores resultados, descubrí que utilizar los textos generados como entrada de los modelos no aportaba consistencia y, además, provocaba errores en la generación de texto, ya que la entrada era demasiado larga y difícil de procesar para los modelos.

Una vez que mi compañero Miguel completó el ajuste fino (fine-tuning), también realicé pruebas con estos mismos prototipos, cambiando el modelo de lenguaje por el que había sido entrenado por mi compañero.

Por último, trabajé en la limpieza de código, agregando comentarios que permitieran que personas ajenas al proyecto pudieran entenderlo, y los traduje al inglés. Debido a la falta de tiempo por la proximidad de las fechas de entrega, evaluamos y

discutimos los resultados obtenidos y determinamos el trabajo futuro necesario para mejorar el proyecto.

Bibliografía

Y así, del mucho leer y del poco dormir, se le secó el celebro de manera que vino a perder el juicio.

Miguel de Cervantes Saavedra

Bezanson, J., Edelman, A., Karpinski, S. y Shah, V. B. Julia: A fresh approach to numerical computing. *SIAM Review*, vol. 59(1), páginas 65–98, 2017.

Bisong, E. y Bisong, E. Google colaboratory. *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners*, páginas 59–64, 2019.

Blinn, J. F. What is a pixel? *IEEE computer graphics and applications*, vol. 25(5), páginas 82–87, 2005.

Dale, R. Gpt-3: What's it good for? *Natural Language Engineering*, vol. 27(1), páginas 113–118, 2021.

Devlin, J., Chang, M.-W., Lee, K. y Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Díaz García, E. y Cabrera Granado, E. Manual de uso de jupyter notebook para aplicaciones docentes. 2018.

Eguíluz Pérez, J. Introducción a javascript. 2012.

González Duque, R. Python para todos. 2011.

Gutiérrez-Fandiño, A., Armengol-Estapé, J., Pàmies, M., Llop-Palao, J., Silveira-Ocampo, J., Carrino, C. P., Armentano-Oller, C., Rodriguez-Penagos, C., Gonzalez-Agirre, A. y Villegas, M. Maria: Spanish language models. *Procesamiento del Lenguaje Natural*, vol. 68(0), páginas 39–60, 2022. ISSN 1989-7553.

Holtzman, A., Buys, J., Forbes, M. y Choi, Y. The curious case of neural text degeneration. *CoRR*, vol. abs/1904.09751, 2019.

KeepCoding. ¿qué es la lematización en spacy? https://keepcoding.io/blog/que-es-la-lematizacion-en-spacy/#:~:text=La%20lematizaci%C3%B3n%20en%20spaCy%2C%20al,a%20su%20ra%C3%ADz%20o%20lema, 2023. Accedido el 22 de mayo de 2023.

Mackay, D. *The fantasy role-playing game: A new performing art*. McFarland, 2017.

Markowitz, D. Transformers: explicamos el modelo detrás de GPT-3, BERT y T5. https://www.ibidemgroup.com/edu/traduccion-automatica-gpt3-bert-t5/, 2023. Accedido el 22 de mayo de 2023.

Midjourney. Página web oficial de midjourney sobre sus modelos de inteligencia artificial. Disponible en https://www.midjourney.com/home/.

OpenAI. Página web oficial de dall-e2. Disponible en https://openai.com/product/dall-e-2.

OpenAI. Página web oficial de chatgpt. Disponible en https://openai.com/blog/chatgpt.

Parlamento Europeo. ¿qué es la inteligencia artificial y cómo se usa? 2021.

Pita, V. y Arribas, V. Text generation with gpt-2. https://www.modeldifferently.com/en/2021/12/generación-de-fake-news-con-gpt-2, 2021. Accedido el 22 de mayo de 2023.

Ripley, B. D. et al. The r project in statistical computing. *MSOR Connections. The newsletter of the LTSN Maths, Stats & OR Network*, vol. 1(1), páginas 23–25, 2001.

Rouhiainen, L. Inteligencia artificial. *Madrid: Alienta Editorial*, 2018.

Schwartz, D. Using discord to facilitate student engagement. 2021.

Serrano, A. V., Subies, G. G., Zamorano, H. M., García, N. A., Samy, D., Sánchez, D. B., Moreno-Sandoval, A., Nieto, M. G. y Jiménez, Á. B. Rigoberta: A state-of-the-art language model for spanish. *ArXiv*, vol. abs/2205.10233, 2022.

Wikipedia. Bert (modelo de lenguaje) — wikipedia, la enciclopedia libre. 2023a. [Internet; descargado 9-mayo-2023].

Wikipedia. Inteligencia artificial — wikipedia, la enciclopedia libre. 2023b. [Internet; descargado 22-mayo-2023].

Wikipedia contributors. Fine-tuning (machine learning) — Wikipedia, the free encyclopedia. 2023a. [Online; accessed 21-May-2023].

Wikipedia contributors. Gpt-2 — Wikipedia, the free encyclopedia. 2023b. [Online; accessed 20-May-2023].

Wikipedia contributors. Transformer (machine learning model) — Wikipedia, the free encyclopedia. 2023c. [Online; accessed 20-May-2023].

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q. y Rush, A. Transformers: State-of-the-art natural language processing. En *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, páginas 38–45. Association for Computational Linguistics, Online, 2020.

Apéndice A

Comparativa entre GPT Neo y GPT2

El presente apéndice contiene las salidas generadas por los modelos de GPT-2 y GPT-Neo del primer prototipo desarrollado. Gracias a estas salidas evaluamos las salidas generadas por los modelos y realizamos comparaciones entre los resultados obtenidos.

- GPT-Neo:

  > Once upon a time in the world of Rinaroland 2 cities, Nesvolland and Nesvolland got into a cruel war between each other. The first battle was fought onthe banksof Lake Trelle-tolte; the second one in which the two men werefighting at once against their enemy's forces under a greatfierce struggle that lasted for some years until theywere finally routed by an army of Nelskorn who had marched from Utterling toSweden with his father in the forest behind him as he went along among themon foot after leaving home the last night on this road.The first day of the third month of the year, Neliswold came up again and met her husband from the shoreup near Hrothgarweg (a town where there is no natural border)who lived about three miles away.She saw the young man coming out towards her, but she did not knowhim or understand what she

- GPT-2:

  > Once upon a time in the world of Rinaroland 2 cities, Nesvolland and Nesvolland got into a cruel war between each other. As King Ogh had been so often the one taking command...there was no place for them either. "No matter how many times I tried to leave him there....Aimee heard him squirm as she walked back toward her hut. The doorbell rang in his ear just before him on either side of the bed. She turned, and ran inside thinking that it might be the end here. At any rate he didnt́ have much choice if Lyrlan came over and demanded his own food or what. He hadnt́ eaten himself since coming home. Ǵoodnight Marnia!h́e said in a low voice, a soft tone, that Kaelic royalty would

> be pleased at seeing them through a distance. He didnt́ like going too far

Los modelos tienen mucho a hablar en primera persona con gente, además de generar mal el apostrofe ', dando falsos negativos al sistema de evaluación de sintaxis. Los modelos presentan una tendencia a utilizar la primera persona y simula una conversación con las personas. Además, en ocasiones, muestran dificultades para generar correctamente el apostrofe én las palabras, lo que puede llevar a falsos negativos en la evaluación del sistema de sintaxis.

- GPT-Neo:

> Once upon a time in the world of Rinaroland 2 cities, Nesvolland and Nesvolland got into a cruel war between each other. The first city was located next to the river named Gjorn-Krul; that is when it had been called Kralnagar by King Jarlś father Gaon (who lived there), but he came back with a band from the south who were called Svermorgen or "the Red,"because they wanted to fight against the king and his people–a fact which he did not know about until they arrived at home at once, for all that he knew they would be killed ifthey found themselves on their way out! So the people went up together as far away as possible sothat they could meet on their own land: this was the place where GaonskroppinḰing Erekir took part during those days; so they got to go overland again. In the”

- GPT-2:

> Once upon a time in the world of Rinaroland 2 cities, Nesvolland and Nesvolland got into a cruel war between each other.The first city was located next to the river named Gennar. The second city is called Gennar. Itś been known as the Great City since it was originally the name for the kingdom of Gennar. “Gennar” means King of all Kings”

GPT-Neo tiende a narrar historias en las que las personas son protagonistas secundarios con mayor frecuencia. Basándonos en esta observación, consideramos que mediante el proceso de fine-tuning podríamos modificar el resultado final para enfocarlo más en el protagonismo de las personas.

- GPT-Neo:

> Once upon a time in the world of Rinaroland 2 cities, Nesvolland and Nesvolland got into a cruel war between each other. The first city was located next to the river named Ashen-Ogle; and when they arrived at that place, Nelskorn had to fight with them for their lives: but he would not let go his sword or any partof it, because, as he saw the battle coming on, he thought that this might be the last chancethat would ever come before him against the enemy or even

> his own people! Andthen, as a man who has been wounded by one side only knows how many wounds have passed throughhis body from one wound overto another, he became very angry about this, so he called out "Nelks"(the son) from behind those two walls which were to hold back allthese enemies; so he came up here among these three men. They said “We are going”

- GPT-2:

> Once upon a time in the world of Rinaroland 2 cities, Nesvolland and Nesvolland got into a cruel war between each other.The first city was located next to the river named Lorkin. The second one had been destroyed by an explosion that took place near the lake. There were many people living there who were trying desperately for survival but it wasn't until they reached the city's borders that they realized what was happening”

Notamos el mismo comportamiento, ambos tienden a finalizar o hablar principalmente sobre personajes específicos del mapa, lo cual no coincide con nuestra intención de tener una introducción más general al inicio. Sin embargo, ocasionalmente GPT-2 logra generar una introducción más adecuada y bien estructurada como la escrita anteriormente.

- GPT-Neo:

> Once upon a time in the world of Rinaroland 2 cities, Nesvolland and Nesvolland got into a cruel war between each other. The first city was located next to the river named Aetna; it had been built by King Villefort from a stone bridge over which he could not cross with his horse or cart for many years before the kingś son was killed on that day when they were fighting together at Dordogne (the next city). He then took possession of the town from his father, and the king built a new castle of the same name, which he called “Nestor”, or Nestil-Othos: Nespresis is a little settlement near here where there are also some small townsfolk who live along this road as well in the days that followed. In the days of the old kings Nescadocs lived among them, but the people of these parts died away after a long while. When”

- GPT-2:

> Once upon a time in the world of Rinaroland 2 cities, Nesvolland and Nesvolland got into a cruel war between each other.The first city was located next to the river named Theros. It had been centuries since their last battle at the end of the war that led them there. Afterward they fought until the city fell back with no hope for its survival or even if it did survive. Nes”

GPT-Neo tiende a repetir el mismo patrón, enfocándose en personas e historias narrativas, lo que no se ajusta a nuestro objetivo de tener una introducción más general sobre el lugar. Por otro lado, GPT-2 intenta generar más frecuentemente una introducción más amplia y descriptiva del sitio. Fue por este estudio por el que decidimos optar por usar GPT-2 y probar a utilizar GPT-Neo en un futuro aplicando el *fine-tuning*

- GPT-Neo:

  > Once upon a time in the world of Rinaroland 2 cities, Nesvolland and Nesvolland got into a cruel war between each other. The first city was located next to the river named Nesvolland, and the second was located on the shore of the lake called Nesvolland Lake. The lake was named after a famous and famous sailor who lived in that city. The lake was named after a famous sailor who lived in that city. In the same way, the lake was named after a famous sailor in the city of Nesvoll"

- GPT-2:

  > Once upon a time in the world of Rinaroland 2 cities, Nesvolland and Nesvolland got into a cruel war between each other.The first city was located next to the river named Rinarland which had been destroyed by an enemy army that invaded the kingdom during World War II. The second city was called Nesvolland where it was claimed as a part of the Empire's plans for the invasion from the north."

- GPT-Neo:

  > Once upon a time in the world of Rinaroland 2 cities, Nesvolland and Nesvolland got into a cruel war between each other. The first city was located next to the river named Elisande, and the second was located to the north. In the middle of the war, the Nelsvollands were attacked by the Nilsvollands. They were killed in the first battle, and the Nelsvollands were killed in the second. After the Nelsvollands had been routed, the Nilsvollands became a"

- GPT-2:

  > Once upon a time in the world of Rinaroland 2 cities, Nesvolland and Nesvolland got into a cruel war between each other.The first city was located next to the river named Nesvolland and several others in smaller towns around the area. The second one was known as Nesvolland. "Nesvolland,"Nesvolland said. It had been there for centuries but it"