Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Generación Automática de textos con GPT-2

Automatic text generation with GPT-2

David Valverde Gómez

La Laguna, 2 de julio de 2021

D. **Jesús Miguel Torres Jorge**, con N.I.F. 43.826.207-Y profesor Contratado Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. **José Demetrio Piñeiro Vera**, con N.I.F. 43.774.048-B profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

”*Generación automática de textos con GPT-2*”

ha sido realizada bajo su dirección por D. **David Valverde Gómez**, con N.I.F. 79.081.228-Y.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 2 de julio de 2021

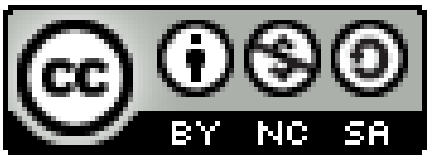
Agradecimientos

A Jesús Miguel Torres Jorge y a José Demetrio Piñeiro Vera, por proporcionarme la idea de este trabajo, así como los recursos y el asesoramiento para llevarlo a cabo.

A mis amigos, compañeros de risas y de penas durante estos años.

Y a mi familia, siempre mi apoyo incondicional en todo lo que hago.

Licencia



Resumen

El Procesamiento del Lenguaje Natural (PLN) es una de las ramas de la Inteligencia Artificial que más atención ha recibido recientemente. Concretamente, su aplicación en la generación automática de textos ha sido foco de gran cantidad de proyectos, entre los que se incluyen los asistentes de los dispositivos móviles, tales como Alexa, Siri o el Asistente de Google.

El objetivo de este proyecto es estudiar una de las herramientas más potentes de este ámbito: el modelo GPT-2 de OpenAI, analizando su funcionamiento, resultados y mejoras, para finalmente construir un pequeño ejemplo 'de juguete' basado en varias series de libros conocidas a nivel mundial.

Para este trabajo se ha utilizado el lenguaje de programación Python, junto con diversas librerías que facilitan la interacción con el modelo.

Palabras clave: GPT-2, OpenAI, Python, Telegram, Inteligencia Artificial, Procesamiento del Lenguaje Natural, Machine Learning, generación automática de textos

Abstract

Natural Language Processing (NLP) is one of the branches of Artificial Intelligence that has received more attention recently. Specifically, it is its application in the automatic text generation that has been the focus of a large number of projects, including assistants for mobile devices, such as Alexa, Siri or the Google Assistant.

The objective of this project is to study one of the most powerful tools in this field: the OpenAI GPT-2 model, analyzing its operation, results and improvements, to finally build a small example based on several well-known series of books.

For the development of this project, the Python programming language has been used, along with some libraries to ease the interaction with the model.

Keywords: GPT-2, OpenAI, Python, Telegram, Artificial Intelligence, Natural Language Processing, Machine Learning, automatic text generation

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Motivación

El Machine Learning (ML) [1] es un campo de la Inteligencia Artificial, relacionado con el Big Data, cuyo objetivo, como su propio nombre indica, es que una máquina sea capaz de aprender, o lo que es lo mismo, detectar patrones en la información.

De esta manera, un modelo informático con una gran cantidad de datos podría entrenarse hasta encontrar un patrón, y así poder incluso predecir datos futuros con una gran fiabilidad.

El Procesamiento del Lenguaje Natural (PLN) es una de las ramas de este campo, que se centra en el estudio del lenguaje, teniendo gran cantidad de aplicaciones como la detección de entidades (Named Entity Recognition o NER), la clasificación de textos o la generación automática de textos. La versatilidad de este campo lo convierte en uno de los focos de atención en la actualidad, destacando su importancia y utilidad en diferentes sectores empresariales.

La aplicación del PLN para la generación automática de textos ha sido el centro de numerosos proyectos durante estos últimos años, tales como los modelos GPT de OpenAI o el prototipo Transformer.

Es la rápida evolución de estos proyectos la que motiva este trabajo, dado que es una tecnología en constante evolución y que en un futuro formará parte del día a día de muchos, como ya ha pasado con los asistentes virtuales tales como Siri, Alexa o Cortana.

1.2. Objetivos

El objetivo principal de este proyecto es el estudio del modelo de generación automática de textos GPT-2 de OpenAI [2], su funcionamiento, resultados y posibles aplicaciones.

Cabe destacar que este modelo se encuentra entrenado actualmente con datos en inglés, por lo que se pretende realizar un entrenamiento sobre el mismo con datos español, de manera que se obtenga un modelo especializado en algún ámbito concreto, siendo en este caso diferentes series de libros famosas entre los jóvenes, tales como Harry Potter o Divergente, todas ellas en español castellano.

Además, para proporcionar un medio de interacción de dichos modelos con el usuario, se implementará un bot de Telegram en el que se pueda indicar una serie de libros concreta y se obtenga como respuesta un texto generado en base a dichos libros.

1.3. Estructura del documento

Este documento está estructurado siguiendo el siguiente esquema:

- **Capítulo 1: Introducción.**

  Se indican las motivaciones y objetivos del proyecto.

- **Capítulo 2: Antecedentes y estado del arte.**

  Estudio de la historia y situación actual de la generación automática de textos, además de los diversos modelos GPT.

- **Capítulo 3. Tecnologías utilizadas.**

  Se indican los recursos empleados en el proyecto, tanto software como hardware.

- **Capítulo 4. Desarrollo.**

  Se explica cada uno de los pasos realizados para la finalización del trabajo.

- **Capítulo 5. Resultados obtenidos.**

  Se muestran los resultados del trabajo realizado.

- **Capítulo 6. Presupuesto.**

  Estimación del coste que supondría el desarrollo de este proyecto.

- **Capítulo 7. Conclusiones y líneas futuras.**

- **Capítulo 8. Summary and Conclusions.**

Capítulo 2

Antecedentes y estado del arte

2.1. Procesamiento del Lenguaje Natural

Durante los últimos años, el PLN ha experimentado una gran evolución, potenciada en gran medida por el desarrollo del Deep Learning [3], un subconjunto de algoritmos de Machine Learning basados en el aprendizaje de múltiples niveles de características, que forman una representación jerárquica.

Originalmente, los problemas de análisis de texto se abordaban utilizando variables del tipo Bag of Words (BoW) [4] utilizadas para realizar un modelado de las palabras basado en diccionarios, sin tener en cuenta el orden de las mismas.

Esta representación se vio sustituida por los llamados word embeddings [5], una forma de modelar las palabras en la que estas se ven representadas por vectores de números reales. De esta manera, las palabras no se encuentran representadas en dimensiones independientes, sino que comparten espacio con otras relacionadas semánticamente (Ej.: perro - gato, rey - reina).

Gracias a este modelado, las redes neuronales recurrentes [6] eran capaces de tener en cuenta no sólo el significado de una palabra, sino también su orden en la frase.

En la Figura 2.1 se puede apreciar el funcionamiento de las redes neuronales recurrentes que utilizan estos embeddings. Siendo h el texto de entrada, x los embeddings correspondientes a cada palabra e y la representación final, se observa cómo cada elemento recibe la representación del elemento anterior, con el fin de utilizarlo para su propio modelado [7].

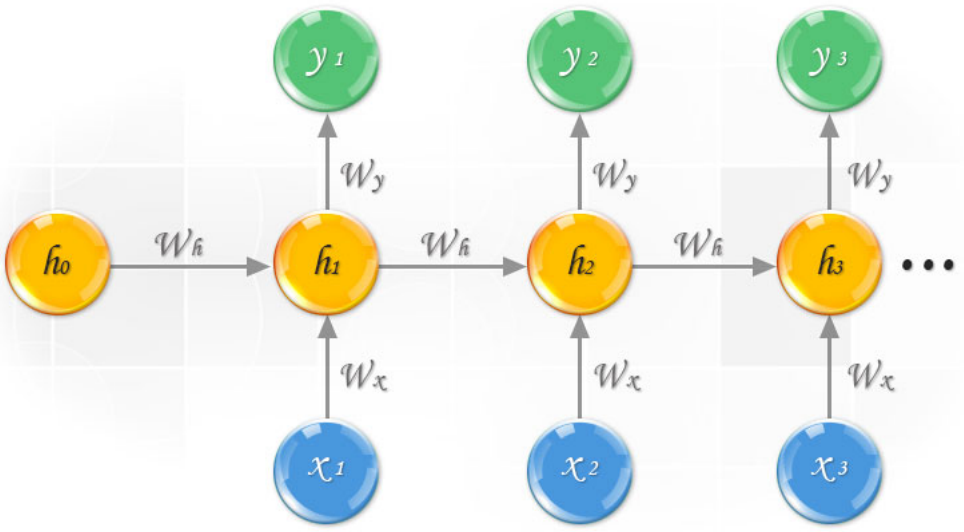


Figura 2.1: Funcionamiento de una Red Neuronal Recurrente utilizando embeddings

2.2. Modelo Transformer

El modelo Transformer fue presentado por Google en el año 2017. Es un modelo del lenguaje [8], es decir, un modelo de Machine Learning cuya finalidad es construir una distribución de probabilidad en la que se asigne a cada palabra su probabilidad de aparición, siempre en función del contexto anterior. Esto permite al modelo predecir con cierta fiabilidad qué palabra debería aparecer en cada momento, lo que da lugar a la generación automática de textos.

La arquitectura de este modelo resultó ser innovadora debido a la implementación de las llamadas capas de atención en las redes neuronales. Estas capas permiten la codificación de cada palabra, de manera que se tenga en consideración el resto de la secuencia, es decir, se introduce el contexto de la frase en la representación matemática del texto. Por este motivo, a los modelos basados en la arquitectura del Transformer se les denomina embeddings contextuales.

Generalmente, se requiere de dos fases para trabajar con el modelo Transformer, así como los modelos basados en el mismo.

- **Pre-training**. En esta fase, el modelo aprende la estructura general del lenguaje, así como un conocimiento base del significado de las palabras. Suele ser un entrenamiento no supervisado con grandes cantidades de datos.

- **Fine-tuning**. Es un entrenamiento utilizado para afinar el modelo, especializándolo en algún ámbito concreto. Se realiza sobre el modelo ya entrenado, es un entrenamiento supervisado y con cantidades de datos no tan grandes .

La gran versatilidad de esta familia de modelos la ha convertido en el foco de las tareas de análisis de texto, siendo la que mejores resultados ha obtenido en el ámbito del PLN. Sin duda, el futuro de estas herramientas está en la reducción de su peso y su costo computacional, siempre sin perder efectividad.

2.3. Modelo GPT

El modelo GPT (Generative Pre-Training Transformer) [9] fue publicado en el año 2018 por OpenAI. Está basado en la arquitectura del modelo Transformer, sobre el que se realiza una secuencia de aprendizaje semi-supervisado: un pre-entrenamiento no supervisado y un afinado supervisado, gracias al cual se puede especializar el modelo en alguna tarea concreta.

Se utilizó un corpus con 7000 libros sin publicar para el entrenamiento previo, el cual ayudó al modelo a aprender sobre la estructura del lenguaje. Este entrenamiento utiliza un total de 117 millones de parámetros para la red neuronal, lo que supone un indicador que permite comparar la eficiencia entre modelos.

No fue necesario mucho esfuerzo para el proceso de afinado, ya que el modelo había aprendido lo suficiente durante el pre-entrenamiento. Esto permitió que GPT superara a otros modelos existentes en 9 de 12 tareas de PLN en las que se realizó la comparación.

Una cualidad de este modelo que llamó la atención de sus desarrolladores es su capacidad de reconocer tareas específicas para las que no ha sido entrenado, y desempeñar un papel aceptable sobre las mismas. Esto recibe el nombre zero-shot [10].

GPT resultó ser un gran avance, ya que en el momento de su publicación, sus resultados superaban a la mayoría de modelos similares existentes. No obstante, fue superado con creces por su sucesor, el modelo GPT-2.

2.4. Modelo GPT-2

Sucesor directo de GPT, el modelo GPT-2 [2] fue publicado en el año 2019 por OpenAI. Contando con un número de parámetros para la red neuronal más de 10 veces mayor que su predecesor (1.500 millones de parámetros), fue pre-entrenado con 40 GigaBytes (GB) de datos, extraídos de las páginas web mejor valoradas de Reddit [11].

El modelo resultó ser tan potente, que sus desarrolladores decidieron no publicar la versión completa de 1.500 millones de parámetros, por miedo a que fuera utilizado con fines ilícitos, tales como la generación de noticias falsas que potencien la desinformación.

Así, la primera versión publicada de este modelo contaba con 117 millones de parámetros. No obstante, a medida que otros modelos similares con mayor potencia eran publicados, OpenAI decidió revelar versiones mejores, hasta finalmente dar a conocer el modelo al completo. Actualmente, existen cuatro versiones publicadas de GPT-2, en función de la cantidad de parámetros que requiere la red neuronal:

- **117M**
- **345M**
- **774M**
- **1.500M**

En las figuras 2.2 y 2.3 se aprecian distintos ejemplos de las aplicaciones del modelo, y viendo la noticia generada en la Figura 2.2, resulta entendible el miedo de sus desarrolladores de que pueda ser utilizado con fines de desinformación

Figura 2.2: Ejemplo de generación automática de texto con GPT-2

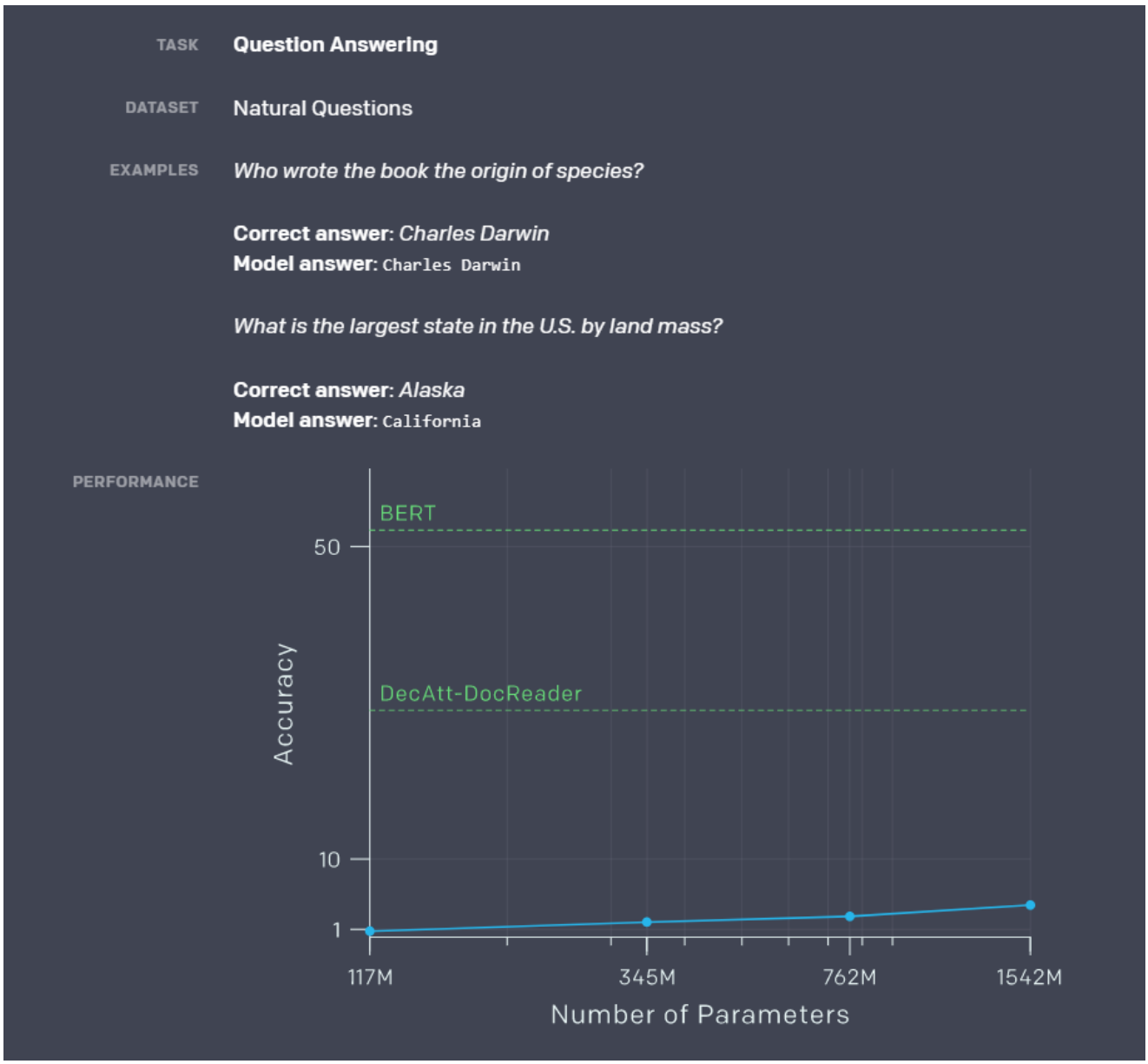


Figura 2.3: Ejemplo de respuestas a preguntas con GPT-2

2.5. Modelo GPT-3

El modelo GPT-3, el más reciente hasta la fecha, fue publicado en el año 2020 por OpenAI a través de una API [12]. De esta manera, la empresa obtiene un mayor control sobre quién, cuándo y para qué se está utilizando este modelo, minimizando el riesgo de su posible uso con fines indebidos.

Si bien GPT-2 contaba con un entrenamiento de 40GB de datos y 1.500 millones de parámetros para la red neuronal, GPT-3 supone una enorme evolución de este modelo, habiendo sido entrenado con 700GB de datos y utilizando 175.000 millones de parámetros.

Aunque resulta tremendamente costoso a nivel computacional por la gran cantidad de datos que maneja, ha proporcionado resultados tan espectaculares como la generación de código a partir de una descripción del mismo, o la generación de expresiones matemáticas en formato LaTeX.

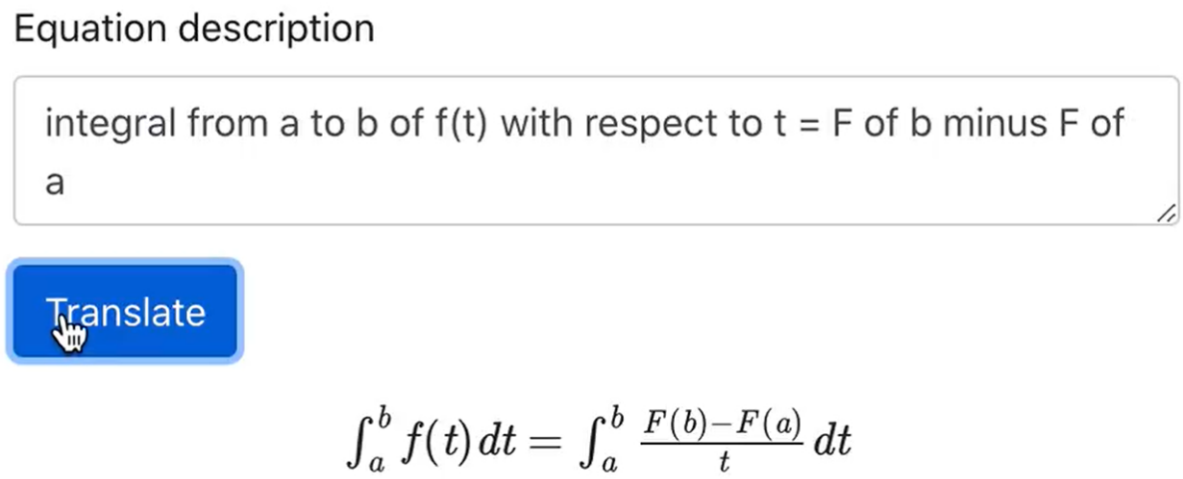


Figura 2.4: Ejemplo 1 de GPT-3

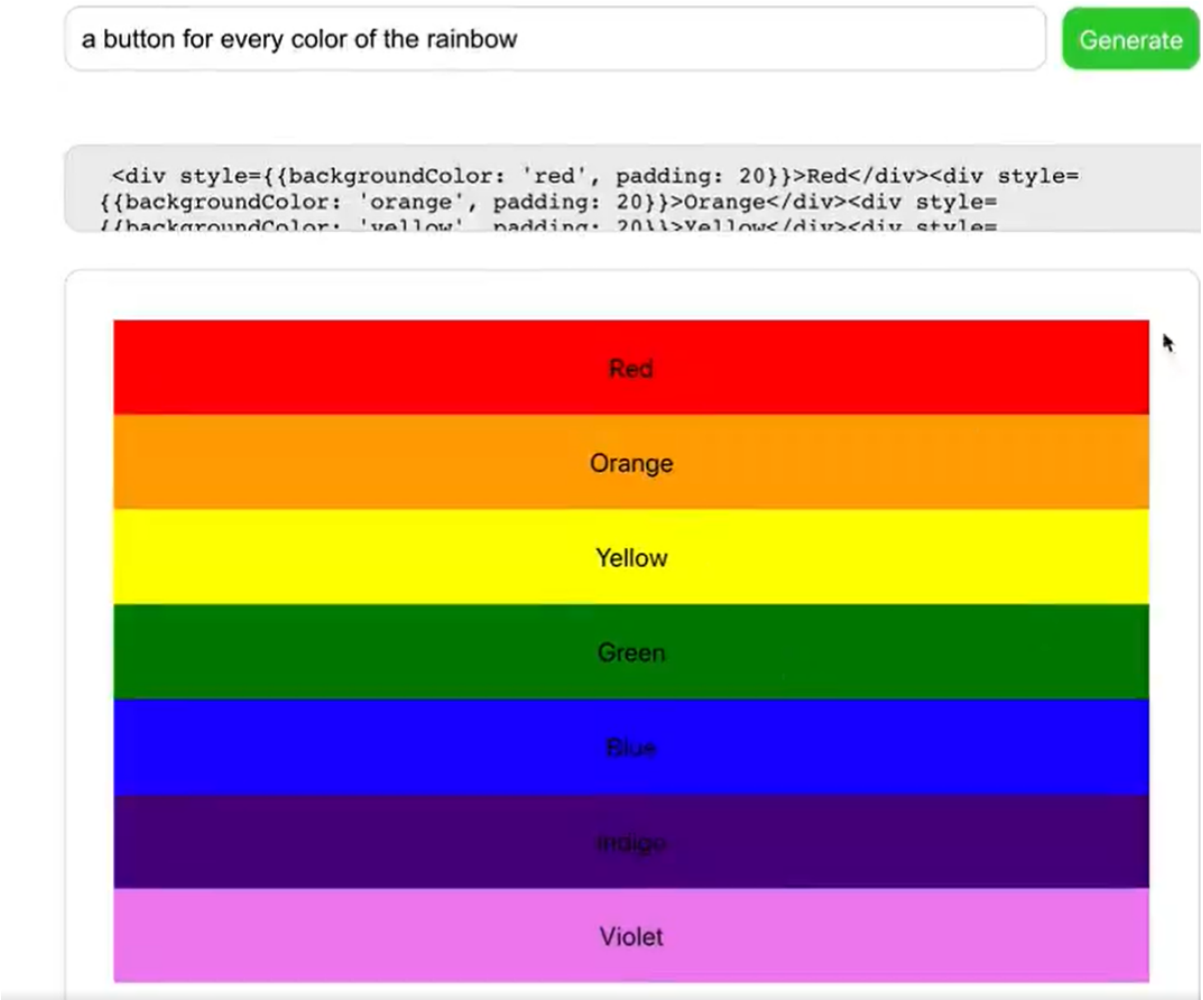


Figura 2.5: Ejemplo 2 de GPT-3

Capítulo 3

Tecnologías utilizadas

3.1. Hardware

La tecnología hardware utilizada para la realización de este proyecto ha sido proporcionada por los tutores mediante un acceso remoto a un ordenador que dispone de diversas GPUs [13]:

- 1 GPU NVIDIA GeForce GTX 980.
- 2 GPUs NVIDIA GeForce RTX 2080.

Estas unidades gráficas han permitido la ejecución de los diversos entrenamientos con una velocidad mayor de la obtenida de haber utilizado una CPU.

Además, la conexión remota a este terminal, así como la búsqueda de información y la redacción de la memoria se han realizado en mi ordenador portátil personal:

- Portátil Lenovo ideapad 530S-151KB con procesador Intel(R) Core (TM) i7-8550U CPU @ 1.80GHz 1.99 GHz y 8GB de RAM.

3.2. Software

En cuanto a las aplicaciones software utilizadas, estas han sido diversas en las diferentes etapas del trabajo:

- Para el desarrollo del proyecto se ha utilizado:
  - El lenguaje de programación **Python** [14]. Versión 3.6.
  - La librería de Python **gpt-2-simple** [15], la cual facilita el acceso al modelo, su ejecución y sus distintos entrenamientos.
  - La librería **python-telegram-bot** [16] para acceder a la configuración del bot de Telegram.
  - Las diversas librerías del sistema tales como **os** o **sys**.
  - Como entorno de desarrollo integrado (IDE) se ha empleado **Visual Studio Code** [17].
  - También se ha requerido de **Anaconda** [18] para establecer un entorno de ejecución en el que instalar las librerías necesarias.
- Para el almacenamiento de los ficheros y modelos:
  - **Google Drive** [19].
  - **GitHub** [20].
- Por último, para la redacción de la memoria:
  - Editor de textos en formato LaTeX **Overleaf** [21].

Capítulo 4

Desarrollo

4.1. Estudio y primeras ideas

La primera fase del proyecto consistió en el estudio y análisis de la herramienta GPT-2: su funcionamiento, estado del arte y posibles aplicaciones. Este aprendizaje se ha visto reflejado en el **Capítulo 2** de esta memoria.

La idea general del proyecto siempre ha sido realizar diversos entrenamientos de afinado (fine-tuning) sobre la versión de 1.500 millones de parámetros del modelo GPT-2, con el objetivo de obtener especializaciones de esta herramienta en el idioma español, ya que el modelo original se encuentra exclusivamente en inglés.

En la búsqueda de una posible aplicación en español para especializar el modelo, surgió la idea de realizar una búsqueda sobre diversos autores célebres de la literatura española, tales como Federico García Lorca o Benito Pérez Galdós, obteniendo así diferentes especializaciones, cada una basada en un autor diferente, y que pudiera generar automáticamente un texto cuyo contenido recordara a uno de estos escritores.

Así pues, se realizó una recopilación de las obras de los siguientes autores:

- **Federico García Lorca** [22].
- **Gustavo Adolfo Bécquer** [23].
- **Benito Pérez Galdós** [24].

Una vez recopiladas las obras, se realizó una limpieza de los datos, dejando únicamente el contenido de cada obra, y se agruparon en ficheros diferenciados por autores, dejando los ficheros listos para realizar el entrenamiento.

4.2. Entrenamientos del modelo

Una vez preparados los ficheros, dio comienzo la fase de entrenamiento. Para ello, se estudió la librería de Python **gpt-2-simple** [15], un módulo gratuito que proporciona diversas funciones para interactuar con el modelo GPT-2. Tras el estudio de su documentación y de varios ejemplos, se implementaron las funciones mostradas a continuación.

La función **download-model** (Figura 4.1) simplemente descarga el modelo indicado (117M, 345M, 774M, 1558M) y lo almacena en ’./models/’, siempre y cuando este no haya sido descargado ya.

```
# Se descarga la versión del modelo indicada, si no se ha descargado ya
def download_model(model_name):
  if not os.path.isdir(os.path.join("models", model_name)):
    print("Downloading " + model_name + " model...")
    gpt2.download_gpt2(model_name=model_name)   # El modelo se guarda en ./models/1558M/
  else:
    print('Model has already been downloaded')
```

Figura 4.1: Función download-model

La función **train** (Figura 4.2) inicia una sesión de TensorFlow [25] sobre la que ejecutar el fine-tuning con los siguientes parámetros:

- **sess**. La sesión de TensorFlow iniciada.
- **run-name**. El nombre con el que se va a guardar el modelo en el directorio ’./checkpoint’.
- **dataset**. La ruta al fichero de datos a utilizar para el entrenamiento.
- **model-name**. La versión del modelo utilizada.
- **steps**. Número de pasos del entrenamiento.
- **restore-from**. Empezar el entrenamiento de nuevo o retomarlo desde el último checkpoint.
- **print-every**. Cada cuántos pasos se debe mostrar el progreso en pantalla.
- **sample-every**. Cada cuántos pasos se debe mostrar un ejemplo de ejecución del modelo.
- **save-every**. Cada cuántos pasos se debe guardar el modelo.

```
# Variables de configuración para el entrenamiento
number_of_steps = 2000          # Número de pasos
restore = 'latest'              # (latest / fresh) Indica si se quiere comenzar el
                                # entrenamiento desde el último punto de control o desde cero
steps_for_print = 10            # Cada cuántos pasos se muestra el progreso
steps_for_sample = 200          # Cada cuántos pasos de muestra un ejemplo de ejecución
steps_for_save = 200            # Cada cuántos pasos se guarda el modelo
# Se entrena el modelo con el fichero y la configuración indicadas
def train(model_name, filename, path_to_file):
  # Se inicia una sesión de tensorflow
  sess = gpt2.start_tf_sess()
  gpt2.finetune(sess,
                run_name=filename,
                dataset=path_to_file,
                model_name=model_name,
                steps=number_of_steps,
                restore_from=restore,
                print_every=steps_for_print,
                sample_every=steps_for_sample,
                save_every=steps_for_save
                )
```

Figura 4.2: Función train

Por último, la función **execute** (Figura 4.3) carga el modelo (checkpoint) indicado y lo ejecuta con los siguientes parámetros:

- **sess**. La sesión de TensorFlow iniciada.
- **run-name**. El nombre del modelo a utilizar (dentro del directorio ’./checkpoint/’.
- **return-as-list**. Indica que se almacene el resultado en una variable, en lugar de mostrarlo directamente por pantalla.
- **prefix**. Texto inicial a partir del cual se desea generar lo demás.

Una vez obtenido el texto generado, este se separa en líneas y palabras, con el fin de seleccionar el número de palabras indicado por la variable **words**, para finalmente retornar la respuesta generada.

```python
# Se ejecuta el modelo
def execute(filename, words, prefix=None):
  sess = gpt2.start_tf_sess()
  # Se carga el checkpoint correspondiente y se ejecuta
  gpt2.load_gpt2(sess, run_name=filename)
  text = gpt2.generate(sess, run_name=filename, return_as_list=True, prefix=prefix)[0]

  # Se separa el texto obtenido en líneas y cada línea en palabras, con el fin de seleccionar las
  # 'words' primeras palabras
  lineas = text.splitlines()
  palabras = []
  for i in range(len(lineas)):
    palabras.append(lineas[i].split())

  # Se almacenan el número de palabras seleccionado, añadiendo el salto de línea correspondiente
  # entre cambios de línea
  word_count = 0
  texto_final = []
  # La variable palabras es un vector donde cada elemento es un vector de palabras relativas a
  # una línea
  for i in range(len(palabras)):
    for j in range(len(palabras[i])):
      texto_final.append(palabras[i][j])
      word_count += 1
      if word_count == words:
        break
    texto_final[len(texto_final)-1] += '\n'
    if word_count == words:
        break

  text = ' '.join(texto_final)

  sess.close()
  return text
```

Figura 4.3: Función execute

El fichero completo que alberga la definición de estas funciones se encuentra en el Apéndice A.1.

Estas funcionalidades son gestionadas por un programa principal (Apéndice A.2) que recibe por línea de comandos el modelo a entrenar o ejecutar, además de una opción que indica si se quiere realizar un entrenamiento, una ejecución o ambos.

Así, gracias a la conexión remota a un ordenador con varias GPUs proporcionada por los tutores de este proyecto, se realizaron los diversos entrenamientos de fine-tuning utilizando las obras recopiladas de los distintos autores.

4.3. Resultados insatisfactorios

Tras un período de entrenamiento, realizando ejecuciones sobre los tres modelos generados, se observó que los resultados, pese a no mostrar problemas con el idioma, no eran satisfactorios, pues carecían de coherencia y presentaban una sintaxis bastante pobre.

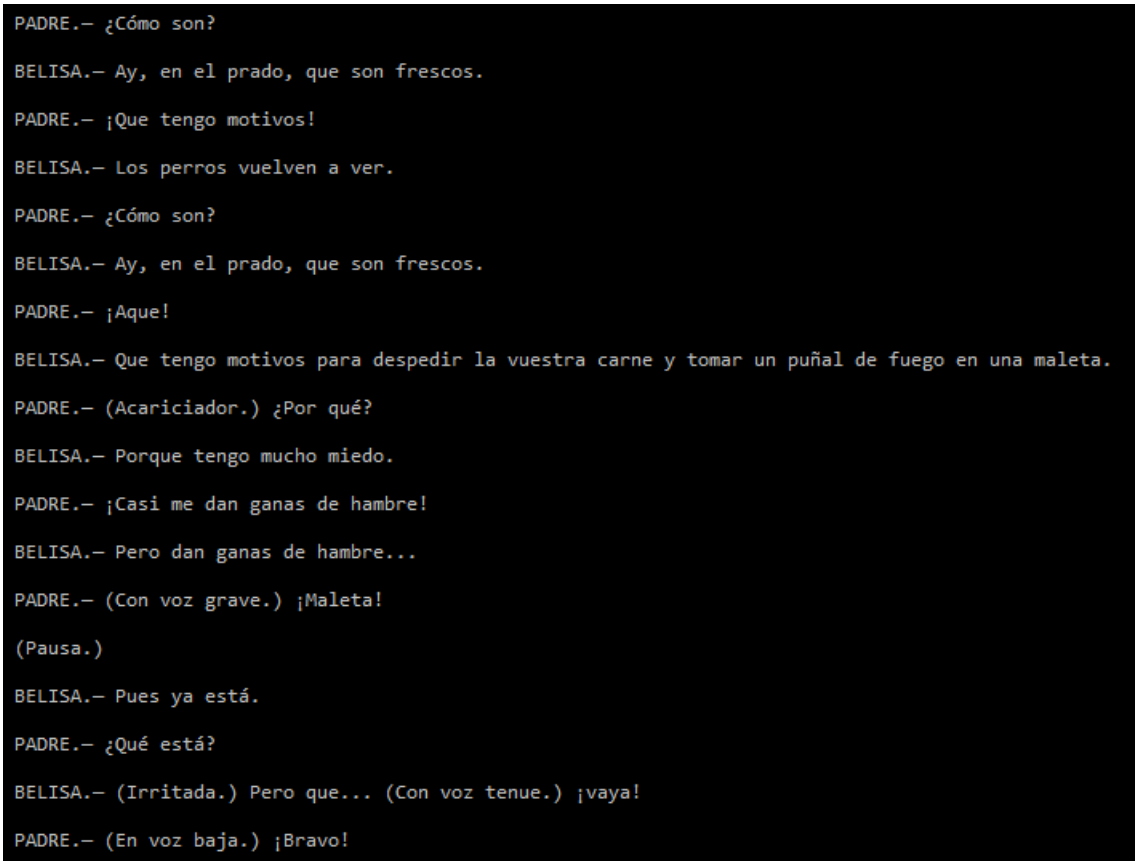


Figura 4.4: Ejemplo de ejecución del modelo de Lorca

Estos resultados se debieron a dos motivos de peso:

- En primer lugar, la cantidad de datos es **muy pequeña**, pues los ficheros de datos apenas superaban 1 MegaByte (MB) de tamaño. El modelo necesita cantidades mucho mayores para proporcionar buenos resultados.
- Además, las obras de un autor son **independientes** entre sí, lo que significa que al juntar todos los escritos en un mismo fichero, el cómputo global carecería de coherencia, que es exactamente lo que está ocurriendo.

Esto no cambió pese a haber incrementado el número de iteraciones del entrenamiento, los resultados seguían siendo similares por las razones anteriores, lo cual impulsó que esta idea fuera desechada y que se debiera buscar una nueva aplicación para el modelo.

4.4. Propuesta final

Los malos resultados obtenidos hasta la fecha impulsaron la búsqueda de un nuevo ámbito en el que especializar la herramienta GPT-2, uno con una mayor cantidad de datos y cuyo contenido mantuviera la coherencia. De esta manera, sin salir del contexto de la literatura, se propuso realizar entrenamientos basados en series de libros famosas, tales como la saga de Harry Potter o de Los Juegos del Hambre.

Pese a que la cantidad de datos no era mucho mayor en comparación con los ejemplos anteriores, ya que los ficheros de datos oscilan entre 3MB y 5MB, se obtuvo la ventaja de que un mismo conjunto de libros mantiene un mismo hilo argumental, por lo que los resultados obtenidos deberían ganar en coherencia.

Con esta idea en mente, se realizó una búsqueda sobre distintas series de libros, se limpiaron los datos, dejando sólo el contenido de las historias, y se agruparon en ficheros por sagas, teniendo como resultado un total de seis ficheros con los que realizar los entrenamientos:

- Harry Potter
- La serie Divergente
- Crepúsculo
- Los Juegos del Hambre
- Las Crónicas de Narnia
- Un mismo fichero con todos los anteriores

Tras entrenar los seis modelos con un total de 2000 iteraciones cada uno, los resultados obtenidos, aunque lejos de ser de buena calidad, parecían mejores que los obtenidos previamente.

```
—¡Lo siento, Harry, querido! ¡Si no me lo puedas, el ministro
puede recordarlo! ¿Qué es lo que en qué consistes eso?
—Esta mañana, ¿y cuándo lo dije? —preguntó Harry, con la boca llena de
chocolate—. Pero no lo dijo. Es muy amable y librándome de los
ojos, pero no deberías sentirmejor lo que yo está conmigo.
—¿Deberías lo que yo está conmigo? —preguntó Ron.
—No. Es una buena idea. Estuvimos de este lugar lo que los hicieron para que yo
estuviera aquí, ¿no?
—No. Es muy bueno.
—Si no lo estuvimos, Harry, eso lo puedo ser.
—No está bien. Mi padre era un gran mago.
—¿Y por qué no?
—No, no pueden ser. Me gustaría que bajara por la escalera de la
cocina.
—¿Cómo puedes llegar a toda velocidad por su atención?
—No, no tengo nada. Mi padre era un gran mago, Harry.
—No puedes que yo estuviera allí a mi padre, ¿verdad? ¿No?
—No puedes ser. Mi padre no era un gran mago y no puede ser.
Se pasaron un momento en silencio.
—No se le habría dado cuenta —dijo Harry mientras miraba a Hermione, que
seguía sin percatarse de la cara de su amiga—. Mi padre era un gran
mago, Harry.
—Si no me hubieras dicho que yo era un gran mago, ¿por qué no?
—No, no lo sé de qué. El gran mago lo sabe. No necesitaba que yo lo
descubrieras.
```

Figura 4.5: Ejemplo de ejecución del modelo de Harry Potter

```
—Katniss —dice Prim en tono mordaz—. Gracias por el cambio de uno de nosotros,
me aseguran que ya era el ejecución que hecho de los juegos.
El aerodeslizador se materializa sobre mí y deja unos cuantos años en el aire.
Después lanza una de las flechas explosivas y me llena el cadáver de
Prim.
—Katniss —me llama Prim—. Lo sé, no, mi madre se lo ha dicho.
Después de unos cuantos años, Prim se ríe.
La voz de Claudius Templesmith retumba en el aerodeslizador; no sé si es de
verdad lo bastante listo para una muerte de tu hermana, de la que mi madre
puede enterarse de todo lo que eres.
—No —me dice—, nos como antes del ataque de noche.
—No, espera, sólo hablaré con ella —le digo, poniéndole los dedos
en los labios para callarlo.
—Pero nos ha dicho que nos ha pedido que vaya a por nosotros, ¿verdad? —
pregunta Prim.
—No es lo que quiero —responde ella, con precaución.
—No, no es… —intenta Delly de nuevo; después seguirándose muy despacio—.
No
sabía lo mucho que aumentar el ese esfuerzo, ¿verdad?
—No, ya no, pero hemos hablado con ella —insisto, poniéndole los dedos
en los labios para callarlo.
—No, no quiero
```

Figura 4.6: Ejemplo de ejecución del modelo de Los Juegos del Hambre

4.5. Bot de Telegram

Una vez obtenidos los modelos finales, afinados con las diferentes series de libros en español, se requería de una aplicación a través de la cual el usuario pudiera interactuar y observar los resultados de las diferentes ejecuciones.

Por este motivo, se ha implementado un bot de Telegram [26], un programa que es ejecutado en en la propia aplicación de mensajería, integrado para que se pueda interactuar con el mismo como si de una conversación normal se tratase.

En concreto, este bot proporcionará dos funcionalidades distintas, dependiendo del tipo de mensaje introducido por el usuario:

- Si el usuario escribe uno de los **comandos** indicados, el bot responderá con un texto generado a partir de una serie de libros u otra, en base al comando concreto introducido. Ej.: /HarryPotter.
- Por otra parte, si se introduce un **mensaje normal**, el bot generará texto a partir de dicho mensaje, de manera que este sirva como 'introducción a la historia'.

Esto ha sido posible gracias a la librería de python **python-telegram-bot** [16], cuya documentación y ejemplos asociados han facilitado la implementación del código para el correcto funcionamiento del bot.

En primer lugar, se definen las funciones a ejecutar cuando se inicia el bot y cuando se introduce el comando /help, las cuales simplemente se limitan a enviar un mensaje informativo sobre el funcionamiento del bot (Figura 4.7).

```python
# Función que retorna un mensaje de bienvenida cuando se recibe el comando /start
def start(update: Update, _: CallbackContext) -> None:
    # Se obtiene el nombre del usuario
    user = update.effective_user
    mensaje_saludo = fr'¡Hola, {user.mention_markdown_v2()}\!'
    update.message.reply_markdown_v2(
        mensaje_saludo
    )

    mensaje_bienvenida = '\nSoy David Valverde, y he creado este bot como complemento p
    update.message.reply_text(mensaje_bienvenida)

# Mensaje de ayuda activado por el comando /help
def help_command(update: Update, _: CallbackContext) -> None:
    mensaje_ayuda = 'Los comandos son: \n-/HarryPotter\n-/LosJuegosDelHambre\n-/Narnia\
    update.message.reply_text(mensaje_ayuda)
```

Figura 4.7: Funciones de bienvenida y de ayuda

A continuación, se definen las funciones que deben ejecutarse cuando se introduce algún comando o cuando se recibe un mensaje normal. Estas funciones consisten básicamente en la generación de texto en base al comando introducido o al mensaje recibido, dependiendo de la decisión del usuario, y en el envío del mensaje generado (Figura 4.8).

```python
# Si se introduce un mensaje normal, se utiliza como prefijo para la ejecución del modelo
def default(update: Update, _: CallbackContext) -> None:
    mensaje = execute('Todos', words=50, prefix=update.message.text)
    update.message.reply_text(mensaje)
    os.execv(sys.executable, ['python3.6'] + sys.argv)


# Esta función se activa cuando se introduce algún comando relativo a los modelos implementados
# y envía como respuesta el resultado de la ejecución del modelo indicado
def generar_texto(update: Update, _: CallbackContext) -> None:
    update.message.reply_text(execute(update.message.text[1:], words=200))
    os.execv(sys.executable, ['python3.6'] + sys.argv)
```

Figura 4.8: Funciones para la generación de texto

Finalmente, se tiene un programa principal que inicializa el bot de Telegram con su token identificativo, para posteriormente asignar los diferentes comandos a su función relacionada y comenzar la ejecución del bot (Figura 4.9).

```python
def main() -> None:
    try:
        # Se accede al bot mediante su token identificativo
        updater = Updater("1877437670:AAH2Dit5e3zhM_3eqKNyL_FtpA0r3LImKYs")
        # 1778530052:AAEIvhMrfRIsEw0LFQzvdKp5zQ4SSE92UAo

        # Objeto usado para asignar las funciones que manejan los comandos
        dispatcher = updater.dispatcher

        # Se definen los comandos permitidos y se relacionan con su correspondiente función
        dispatcher.add_handler(CommandHandler("start", start))
        dispatcher.add_handler(CommandHandler("help", help_command))
        dispatcher.add_handler(CommandHandler("Divergente", generar_texto))
        dispatcher.add_handler(CommandHandler("HarryPotter", generar_texto))
        dispatcher.add_handler(CommandHandler("LosJuegosDelHambre", generar_texto))
        dispatcher.add_handler(CommandHandler("Narnia", generar_texto))
        dispatcher.add_handler(CommandHandler("Crepusculo", generar_texto))
        dispatcher.add_handler(CommandHandler("Todos", generar_texto))


        # Función asociada con los mensajes normales que no son comandos
        dispatcher.add_handler(MessageHandler(Filters.text & ~Filters.command, default))

        # Se inicia el bot
        updater.start_polling()

        # El bot se ejecuta hasta que se interrumpa manualmente la ejecución con Ctrl+C
        updater.idle()

    except Exception as e:
        print(e)
```

Figura 4.9: Programa principal

El código completo se encuentra en el Apéndice A.3.

Capítulo 5

Resultados obtenidos

El resultado final de este Trabajo de Fin de Grado se ve reflejado en la creación de un bot de Telegram completamente operativo, en el que el usuario puede indicar qué tipo de texto desea que se genere de manera automática, bien a través de comandos concretos o bien mediante mensajes normales.

El bot resultante proporciona mensajes de bienvenida y de ayuda para orientar al usuario sobre su uso (Figuras 5.1 y 5.2).

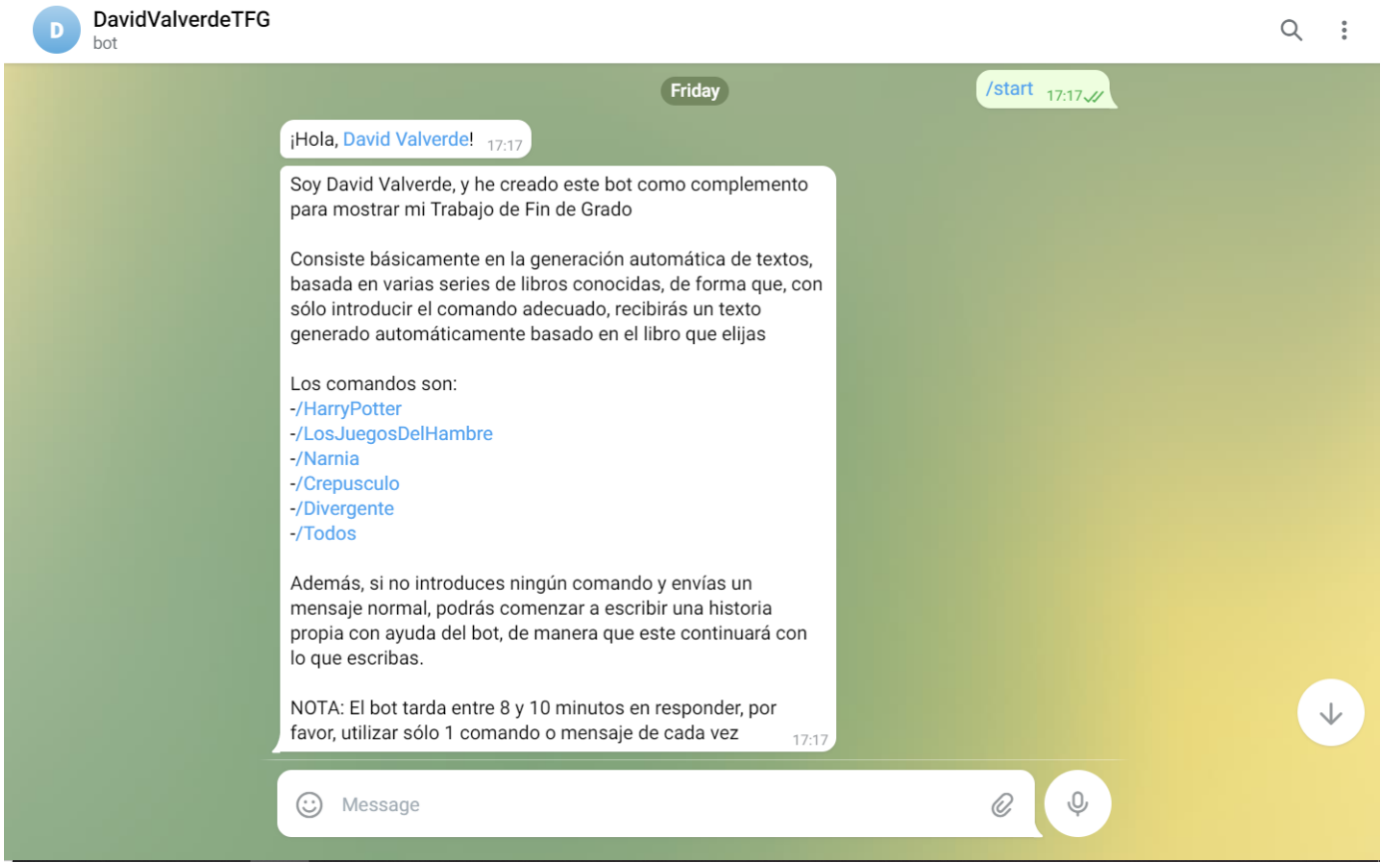


Figura 5.1: Mensaje de bienvenida

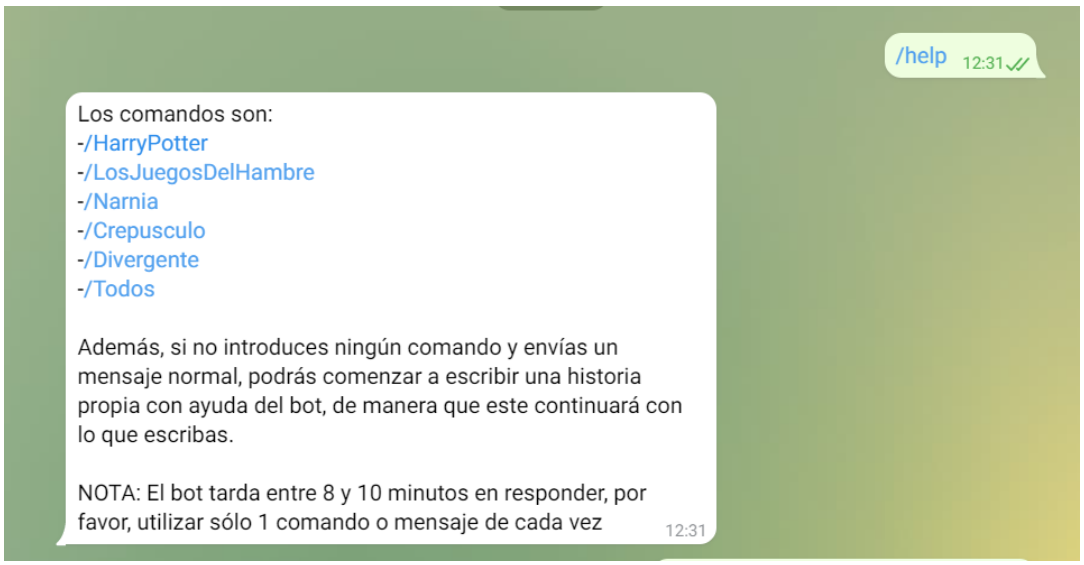


Figura 5.2: Mensaje de ayuda

Los resultados del modelo GPT-2 no han sido del todo satisfactorios, debido en gran parte a la escasez de datos y a la falta de recursos. No obstante, se han obtenido resultados mejores de los obtenidos con la propuesta original, encontrando ciertos fragmentos en el texto que podrían considerarse aceptables (Figuras 5.3 y 5.4).

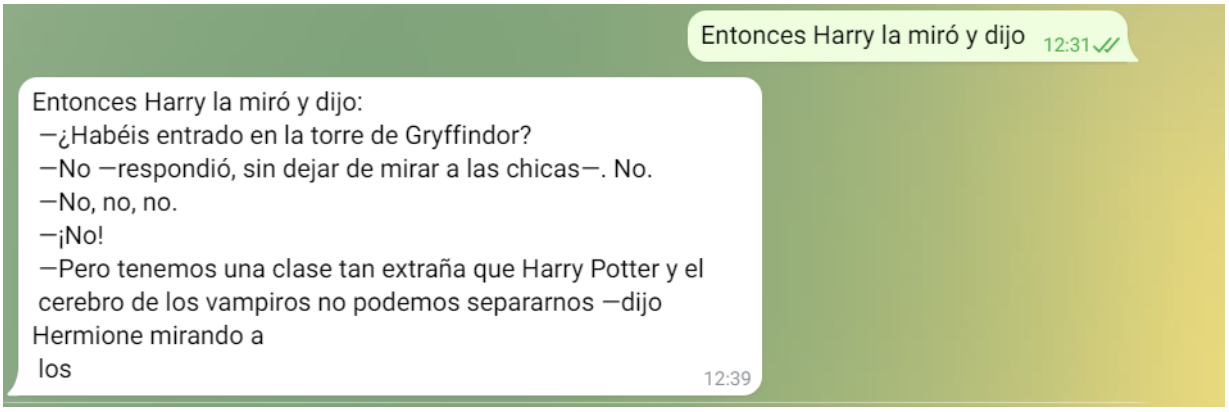


Figura 5.3: Ejemplo de resultado de introducción de texto por el usuario

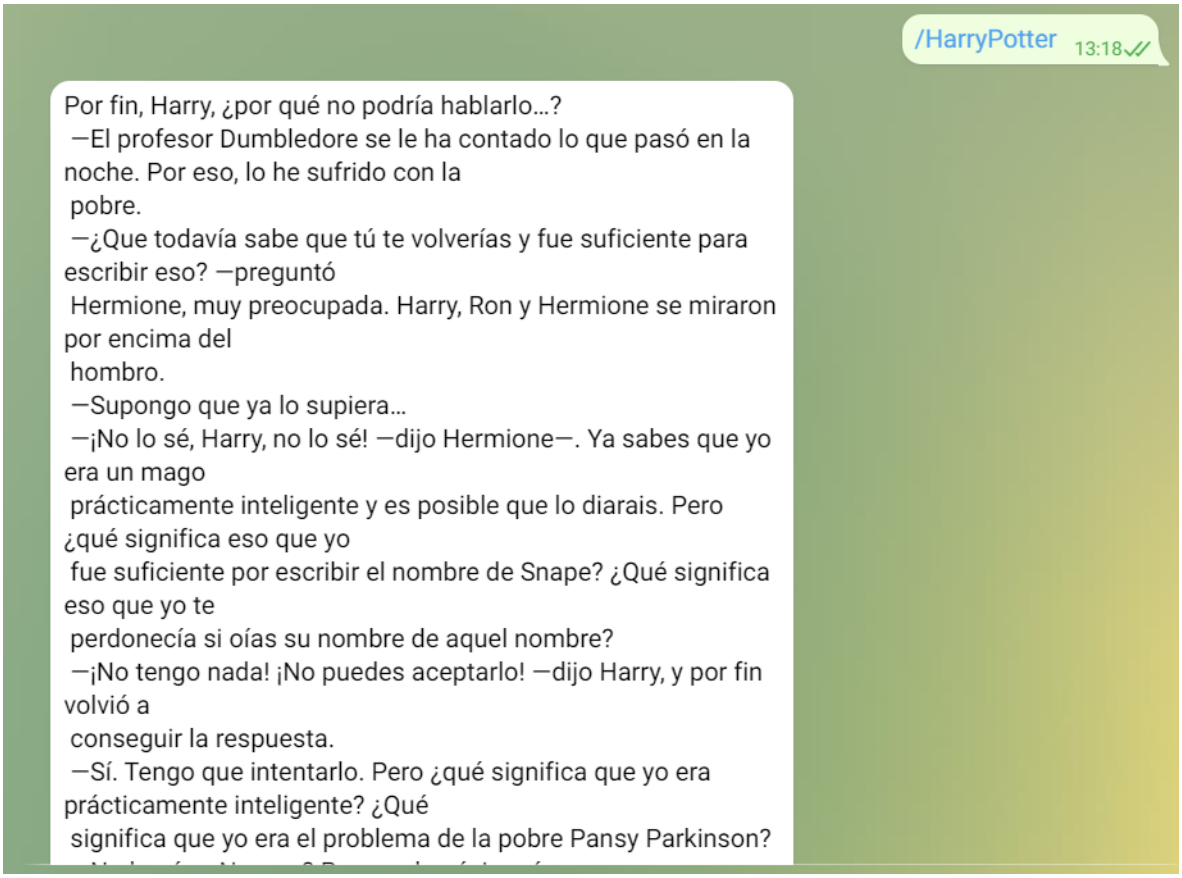


Figura 5.4: Ejemplo de resultado del modelo de Harry Potter

Capítulo 6

Presupuesto

En la Tabla 6.1 se detallan las tareas realizadas y el tiempo invertido en cada una en horas, además de los recursos necesarios para llevar a cabo este proyecto, todo ello acompañado de su precio correspondiente.

El salario medio de un Ingeniero Informático es de unos 36.500,00 euros brutos anuales, alrededor de 3.000,00 euros brutos mensuales [27]. Según esto, y teniendo en cuenta que un mes tiene 20 días laborales y suponiendo una jornada de 8 horas, el coste de cada hora de trabajo es de 19,00 euros brutos.

Todo esto, aplicado a las horas de trabajo dedicadas a cada fase del proyecto, y sumado a los recursos necesarios para su realización, ofrece el presupuesto total en bruto del proyecto.

Descripción	**Cantidad**	**Precio (€)**	**Total (€)**
Planificación del proyecto	10	19,00	190,00
Documentación y estudio sobre la herramienta	80	19,00	1.520,00
Recopilación de obras de autores españoles	15	19,00	285,00
Primer entrenamiento de modelos	20	19,00	380,00
Recopilación de sagas de libros	5	19,00	95,00
Segundo entrenamiento de modelos	50	19,00	950,00
Implementación del Bot de Telegram	20	19,00	380,00
Ordenador portátil	1	599,00	599,00
GPU NVIDIA GeForce GTX 980	1	475,00	475,00
GPU NVIDIA GeForce RTX 2080	2	1289,00	2578,00
		TOTAL:	7.233,00

Tabla 6.1: Presupuesto del proyecto

Capítulo 7

Conclusiones y líneas futuras

La generación automática de textos es, sin duda, una de las ramas más versátiles del campo del Procesamiento del Lenguaje Natural. Su gran número de posibles aplicaciones la sitúa en el centro de atención de numerosas empresas, que ya han desarrollado proyectos tan potentes como los modelos GPT, y que ciertamente no dejará de mejorar.

En este trabajo se han estudiado detenidamente los modelos GPT-2 y GPT-3, pudiendo comprobar en primera persona sus resultados y posibles usos. Algunos tan espectaculares como la generación de contenido web a partir de una simple descripción.

No obstante, ha quedado demostrado que se requieren de inmensas cantidades de datos para poder obtener buenos resultados, ya que se ha observado que los textos generados tras entrenamientos con pocos MegaBytes de datos no son del todo coherentes y no están bien estructurados, por lo que pese al gran potencial de la herramienta, queda limitada al conjunto de datos que pueda proporcionar el usuario.

Otro aspecto a destacar es la gran cantidad de recursos que requieren los entrenamientos, llegando a durar hasta diez horas utilizando tres tarjetas gráficas de alta calidad.

El bot de telegram implementado tiene un enorme margen de mejora, pues se pueden preparar modelos con mayores cantidades de datos, que otorguen una mayor coherencia a los resultados, pudiendo llegar a tener textos que parezcan fragmentos de libros. Con mejores recursos, también sería posible mejorar la velocidad de respuesta, pasando de durar en torno a 10 minutos a ser casi instantáneo, o incluso añadir funcionalidades a la aplicación, tales como la posibilidad de entablar una conversación con el programa.

Capítulo 8

Summary and Conclusions

Automatic text generation is, with no doubt, one of the most versatile branches in the field of Natural Language Processing. It is its large number of possible applications that places it in the focus of many companies, which have already developed such powerful projects as GPT models, that will certainly not stop improving.

In this project, models GPT-2 and GPT-3 have been studied, and their results and possible uses have been proved. Some of them as spectacular as the generation of web content from just a simple description.

Nevertheless, it has been demonstrated that huge amounts of data are required in order to obtain good results, as texts generated with a model that has been trained with a few MegaBytes of data are not neither coherents or well-structured. So, besides the great potential of this tool, it is limited to the set of data that the user can provide.

Another aspect to highlight is the large sum of resources that trainings require, having trainings to last up to ten hours, even using three high-quality graphic units.

The Telegram Bot of this project has room for improvement, as other models might be trained with greater sets of data, making the texts more coherent, to the point that the generated text could seem like an actual part of a book. Using better resources, it would also be possible to improve the response speed, going from lasting around 10 minutes to being almost instantaneous, or even more functionalities could be added to the application, such as the chance of begin a conversation with the bot.

Apéndice A

Código utilizado

A.1. Funciones para GPT-2

```
# David Valverde Gómez
# Ingeniería Informática - ULL
# alu0101100296@ull.edu.es
# Trabajo de Fin de Grado
# Generación automática de textos utilizando el modelo GPT-2 de OpenAI

# Fichero con las funciones necesarias para la descarga, entrenamiento
# y ejecución del modelo

# Se indican las GPUs a utilizar durante la ejecución
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "0, 1, 2"

# El módulo gpt_2_simple proporciona muchas facilidades al interactuar con GPT-2
import gpt_2_simple as gpt2

# Variables de configuración para el entrenamiento
number_of_steps = 2000        # Número de pasos
restore = 'latest'            # (latest / fresh) Indica si se quiere comenzar el
                              # entrenamiento
                              # desde el último punto de control o desde cero
steps_for_print = 10          # Cada cuántos pasos se muestra el progreso
steps_for_sample = 200        # Cada cuántos pasos de muestra un ejemplo de ejecución
steps_for_save = 200          # Cada cuántos pasos se guarda el modelo


# Se descarga la versión del modelo indicada, si no se ha descargado ya
def download_model(model_name):
  if not os.path.isdir(os.path.join("models", model_name)):
    print("Downloading " + model_name + " model...")
    gpt2.download_gpt2(model_name=model_name)   # El modelo se guarda en ./models/1558M/
  else:
    print('Model has already been downloaded')
```

```
# Se entrena el modelo con el fichero y la configuración indicadas
def train(model_name, filename, path_to_file):
  # Se inicia una sesión de tensorflow
  sess = gpt2.start_tf_sess()
  gpt2.finetune(sess,
                run_name=filename,
                dataset=path_to_file,
                model_name=model_name,
                steps=number_of_steps,
                restore_from=restore,
                print_every=steps_for_print,
                sample_every=steps_for_sample,
                save_every=steps_for_save
                )

# Se ejecuta el modelo
def execute(filename, words, prefix=None):
  sess = gpt2.start_tf_sess()
  # Se carga el checkpoint correspondiente y se ejecuta
  gpt2.load_gpt2(sess, run_name=filename)
  text = gpt2.generate(sess, run_name=filename, return_as_list=True, prefix=prefix)[0]

  # Se separa el texto obtenido en líneas y cada línea en palabras, con el fin de
  # seleccionar las 'words' primeras palabras
  lineas = text.splitlines()
  palabras = []
  for i in range(len(lineas)):
    palabras.append(lineas[i].split())

  # Se almacenan el número de palabras seleccionado, añadiendo el salto de línea
  # correspondiente entre cambios de línea
  word_count = 0
  texto_final = []
  # La variable palabras es un vector donde cada elemento es un vector de palabras
  # relativas a una línea
  for i in range(len(palabras)):
    for j in range(len(palabras[i])):
      texto_final.append(palabras[i][j])
      word_count += 1
      if word_count == words:
        break
    texto_final[len(texto_final)-1] += '\n'
    if word_count == words:
        break

  text = ' '.join(texto_final)

  sess.close()
  return text
```

A.2. Programa Principal

```
# David Valverde Gómez
# Ingeniería Informática - ULL
# alu0101100296@ull.edu.es
# Trabajo de Fin de Grado
# Generación automática de textos utilizando el modelo GPT-2 de OpenAI

# Este fichero contiene la ejecución para descargar la versión indicada
# del modelo GPT-2, realizar un entrenamiento sobre dicho modelo con ficheros de datos
# pasados por argumento y ejecutar modelos ya entrenados

# Se indican las GPUs a utilizar durante la ejecución
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "0, 1, 2"

# El módulo gpt_2_simple proporciona muchas facilidades al interactuar con GPT-2
import gpt_2_simple as gpt2
import sys

# Se importan las funciones necesarias
from main_functions import *

# Versión del modelo a descargar
model_name = "1558M"

# Directorios con los archivos de entrenamiento y los modelos ya entrenados (checkpoints)
files_directory = 'ficheros_datos/'
checkpoints_directory = 'checkpoint/'

# Archivo concreto a utilizar para el entrenamiento
filename = sys.argv[1]
path_to_file = files_directory + filename + '.txt'

# El modelo va a tener el mismo nombre que el archivo utilizado para su entrenamiento
#checkpoint = checkpoints_directory + filename

# Indicador de si se desea entrenar o ejecutar el modelo
train_or_execute = int(sys.argv[2]) # 1 - Entrenar, 2 - Ejecutar, 0 - Ambos

# Se comprueba que los argumentos introducidos son los indicados
def check_arguments():
  # Se revisan las opciones de entrenamiento, ejecución o ambas
  if train_or_execute != 0 and train_or_execute != 1 and train_or_execute != 2:
    raise Exception ('\nSe requiere un segundo argumento para indicar si se desea
    entrenar o ejecutar el modelo:\n1 - Entrenar\n2 - Ejecutar\0- Ambos')

  # Si se va a entrenar el modelo, se verifica que el archivo indicado se encuentre en el
  # directorio correspondiente
  elif train_or_execute == 0 or train_or_execute == 1:
    files_for_training = os.listdir(files_directory)
    if filename + '.txt' not in files_for_training:
      raise Exception ('\nError. Se intenta entrenar el modelo con un archivo que no
      existe en ' + files_directory)
```

```
  # Si se va a ejecutar, se revisa que se tenga el checkpoint indicado
  else:
    checkpoints = os.listdir(checkpoints_directory)
    if filename not in checkpoints:
      raise Exception('\nError. Se intenta ejecutar un modelo que no existe en ' +
      checkpoints_directory)

# Función principal
def main():
  try:
    check_arguments()
    download_model(model_name)

    if train_or_execute == 0 or train_or_execute == 1:
      train(model_name, filename, path_to_file)
    if train_or_execute == 0 or train_or_execute == 2:
      text = execute(filename)
      print(text)

  except Exception as e:
    print(str(e))


if __name__ == '__main__':
  main()
```

A.3. Bot de Telegram

```
# David Valverde Gómez
# Ingeniería Informática - ULL
# alu0101100296@ull.edu.es
# Trabajo de Fin de Grado
# Generación automática de textos utilizando el modelo GPT-2 de OpenAI

# Este fichero contiene la configuración del bot de Telegram diseñado
# para mostrar los resultados de los diferentes modelos entrenados

# La idea es que, cuando se introduzca un comando determinado, el bot responda
# con un texto generado por el modelo indicado según dicho comando.

# Cuando no se introduzca ningún comando, sino se envíe un mensaje normal, ese mensaje
# se utilice como parámetro para generar texto a partir del mismo, utilizando el modelo
# entrenado por todos los ficheros de datos, de manera que se responda con un mensaje
# que continúe lo escrito por el usuario.

import os
import sys

# Se importan las librerías de configuración de Telegram
import logging
from telegram import Update, ForceReply
from telegram.ext import Updater, CommandHandler, MessageHandler, Filters,
CallbackContext

# Se importa la función para ejecutar los modelos
from main_functions import execute
# Activar logging
logging.basicConfig(
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s', level=logging.INFO
)

logger = logging.getLogger(__name__)

# Se definen las funciones que manejan los comandos
# Función que retorna un mensaje de bienvenida cuando se recibe el comando /start
def start(update: Update, _: CallbackContext) -> None:
    # Se obtiene el nombre del usuario
    user = update.effective_user
    mensaje_saludo = fr'¡Hola, {user.mention_markdown_v2()}\!'
    update.message.reply_markdown_v2(
        mensaje_saludo
    )
    mensaje_bienvenida = '\nSoy David Valverde, y he creado este bot como complemento
    para mostrar mi Trabajo de Fin de Grado\n\nConsiste básicamente en la generación
    automática de textos, basada en varias series de libros conocidas, de forma que,
    con sólo introducir el comando adecuado, recibirás un texto generado automáticamente
    basado en el libro que elijas \n\nLos comandos son: \n-/HarryPotter
    \n-/LosJuegosDelHambre\n-/Narnia \n-/Crepusculo \n-/Divergente\n-/Todos\n
    \nAdemás, si no introduces ningún comando y envías un mensaje normal, podrás
    comenzar a escribir una historia propia con ayuda del bot, de manera que este
    continuará con lo que escribas.\n\nNOTA: El bot tarda entre 8 y 10 minutos en
    responder, por favor, utilizar sólo 1 comando o mensaje de cada vez'
```

```
    update.message.reply_text(mensaje_bienvenida)

# Mensaje de ayuda activado por el comando /help
def help_command(update: Update, _: CallbackContext) -> None:
    mensaje_ayuda = 'Los comandos son: \n-/HarryPotter\n-/LosJuegosDelHambre\n-/Narnia
    \n-/Crepusculo\n-/Divergente\n-/Todos\n
    \nAdemás, si no introduces ningún comando y envías un mensaje normal, podrás
    comenzar a escribir una historia propia con ayuda del bot, de manera que este
    continuará con lo que escribas.\n\nNOTA: El bot tarda entre 8 y 10 minutos
    en responder, por favor, utilizar sólo 1 comando o mensaje de cada vez'

    update.message.reply_text(mensaje_ayuda)

# Si se introduce un mensaje normal, se utiliza como prefijo para la
# ejecución del modelo
def default(update: Update, _: CallbackContext) -> None:
    mensaje = execute('Todos', words=50, prefix=update.message.text)
    update.message.reply_text(mensaje)
    os.execv(sys.executable, ['python3.6'] + sys.argv)

# Esta función se activa cuando se introduce algún comando relativo a los modelos
# implementados y envía como respuesta el resultado de la ejecución del modelo indicado
def generar_texto(update: Update, _: CallbackContext) -> None:
    update.message.reply_text(execute(update.message.text[1:], words=200))
    os.execv(sys.executable, ['python3.6'] + sys.argv)

def main() -> None:
    try:
        # Se accede al bot mediante su token identificativo
        updater = Updater("1877437670:AAH2Dit5e3zhM_3eqKNyL_FtpA0r3LImKYs")
        # 1778530052:AAEIvhMrfRIsEw0LFQzvdKp5zQ4SSE92UAo

        # Objeto usado para asignar las funciones que manejan los comandos
        dispatcher = updater.dispatcher

        # Se definen los comandos permitidos y se relacionan con su
        # correspondiente función
        dispatcher.add_handler(CommandHandler("start", start))
        dispatcher.add_handler(CommandHandler("help", help_command))
        dispatcher.add_handler(CommandHandler("Divergente", generar_texto))
        dispatcher.add_handler(CommandHandler("HarryPotter", generar_texto))
        dispatcher.add_handler(CommandHandler("LosJuegosDelHambre", generar_texto))
        dispatcher.add_handler(CommandHandler("Narnia", generar_texto))
        dispatcher.add_handler(CommandHandler("Crepusculo", generar_texto))
        dispatcher.add_handler(CommandHandler("Todos", generar_texto))

        # Función asociada con los mensajes normales que no son comandos
        dispatcher.add_handler(MessageHandler(Filters.text & ~Filters.command, default))
        # Se inicia el bot
        updater.start_polling()
        # El bot se ejecuta hasta que se interrumpa manualmente la ejecución con Ctrl+C
        updater.idle()

    except Exception as e:
        print(e)

if __name__ == '__main__':
    main()
```

Bibliografía

[1] Machine learning. https://cleverdata.io/que-es-machine-learning-big-data/. ONLINE Accessed: 2021-06-26.

[2] Alec Radford, Jeffrey Wu, Dario Amodei, Daniela Amodei, Jack Clark, Miles Brundage, and Ilya Sutskever. Better language models and their implications. 2019.

[3] Deep learning. https://es.wikipedia.org/wiki/Aprendizaje_profundo. ONLINE Accessed: 2021-06-26.

[4] Bag of words. https://es.wikipedia.org/wiki/Modelo_bolsa_de_palabras. ONLINE Accessed: 2021-06-26.

[5] Word embedding. https://es.wikipedia.org/wiki/Word_embedding. ONLINE Accessed: 2021-06-26.

[6] Redes neuronales recurrentes. https://es.wikipedia.org/wiki/Red_neuronal_recurrente. ONLINE Accessed: 2021-06-26.

[7] Alejandro Vaca. Transformers en procesamiento del lenguaje natural. 2020.

[8] Modelo del lenguaje. https://es.wikipedia.org/wiki/Modelaci%C3%B3n_del_lenguaje. ONLINE Accessed: 2021-06-26.

[9] Alec Radford. Improving language understanding with unsupervised learning. 2018.

[10] Priya Shree. The journey of open ai gpt models. 2020.

[11] Reddit. https://www.reddit.com/. ONLINE Accessed: 2021-06-26.

[12] Greg Brockman, Mira Murati, and Peter Welinder. Openai api. 2020.

[13] Gpu. https://es.wikipedia.org/wiki/Unidad_de_procesamiento_gr%C3%A1fico. ONLINE Accessed: 2021-06-26.

[14] Python. https://www.python.org/. ONLINE Accessed: 2021-06-26.

[15] Librería gpt-2-simple. https://pypi.org/project/gpt-2-simple/. ONLINE Accessed: 2021-06-26.

[16] Librería python-telegram-bot. https://python-telegram-bot.org/. ONLINE Accessed: 2021-06-26.

[17] Visual studio code. https://code.visualstudio.com/. ONLINE Accessed: 2021-06-26.

[18] Anaconda. https://www.anaconda.com/. ONLINE Accessed: 2021-06-26.

[19] Google drive. https://www.google.com/intl/es_es/drive/. ONLINE Accessed: 2021-06-26.

[20] Github. https://github.com/. ONLINE Accessed: 2021-06-26.

[21] Overleaf. https://es.overleaf.com. ONLINE Accessed: 2021-06-26.

[22] Obras de federico garcía lorca. https://federicogarcialorca.net/index.htm. ONLINE Accessed: 2021-06-28.

[23] Obras de gustavo adolfo bécquer. https://es.wikisource.org/wiki/Categor%C3%ADa:Obras_de_Gustavo_Adolfo_B%C3%A9cquer. ONLINE Accessed: 2021-06-28.

[24] Obras de benito pérez galdós. https://es.wikisource.org/wiki/Categor%C3%ADa:Novelas_de_Benito_P%C3%A9rez_Gald%C3%B3s. ONLINE Accessed: 2021-06-28.

[25] Tensorflow. https://www.tensorflow.org/?hl=es-419. ONLINE Accessed: 2021-06-29.

[26] YÚBAL FERNÁNDEZ. Bots de telegram: qué son, cómo funcionan y 17 recomendados para empezar. 2020.

[27] Sueldo del ingeniero informático en españa. https://www.jobted.es/salario/ingeniero-inform%C3%A1tico. ONLINE Accessed: 2021-06-29.