Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Deep Learning en interfaces conversacionales

Deep Learning on Conversational Interfaces

Juan Siverio Rojas

La Laguna, 14 de junio de 2022

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 Piñero 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:

"Deep Learning en interfaces conversacionales"

ha sido realizada bajo su dirección por **D. Juan Siverio Rojas**,

con N.I.F. 78.556.096-X.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 14 de junio de 2022

Agradecimientos

A mi familia, especialmente a mi esposa.

Licencia

© Esta obra está bajo una licencia de Creative Commons Reconocimiento 4.0 Internacional.

Resumen

El objetivo de este TFG es la creación de un asistente virtual, tipo chatbot, utilizando de base, uno de los modelos del lenguaje más avanzados en la actualidad, llamado GPT (Generative Pre-Trained Transformer) por sus siglas en inglés.

Se usa este chatbot, para poner a prueba este Modelo del Lenguaje en su tercera y última versión, GPT-3, recopilando información sobre sus fortalezas y debilidades.

Dispone de una interfaz de usuario multiplataforma desde la que se puede interactuar con GPT-3, y también con Twitter, ya que este último es la base de conocimiento utilizada para su entrenamiento, obteniendo la información desde las cuentas de esta plataforma que se especifiquen. Además, se puede ajustar y configurar con la información que se desee, permitiendo especializarlo en diferentes tareas.

Palabras clave: *machine learning, deep learning, transformer, gpt-3, chatbot, twitter*

Abstract

The objective of this TFG is the creation of a virtual assistant, *chatbot* type, based on one of the most advanced language models today, called GPT (Generative Pre-Trained Transformer).

This *chatbot* is used to test this Language Model in its third and latest version, GPT-3, gathering information about its strengths and weaknesses.

It has a multiplatform user interface, from which anyone can interact with GPT-3, and also with Twitter, since the latter is the knowledge base used for training. Information is obtained from specified accounts of this social network. In addition, it can be adjusted and configured with the information you want, allowing it to be specialized in different tasks

Keywords: *machine learning, deep learning, transformer, gpt-3, chatbot, twitter*

Índice General

Capítulo 1 Introducción 9
1.1 Motivación 9
1.2 Objetivos 10
1.3 Estado del arte 10
1.4 Estructura del documento 11
Capítulo 2 Fundamentos Teóricos 12
2.1 NLP: Embeddings y Transformers 12
2.1.1 Embeddings 12
2.1.2 Word Embeddings 13
2.1.3 Transformers 14
2.2 GPT-3 15
2.1.1 Glosario de términos 15
2.1.2 Modos de funcionamiento 17
2.1.2.1 Modelos Instruct 17
2.1.2.2 Modelos Base GPT-3 18
2.1.2.3 Carga de Ficheros 18
2.1.3 Límite Prompts + Completions 19
2.1.4 Monetización 20
2.1.5 Fine-tuning 20
Capítulo 3 Implementación 23
3.1 Aplicación 23
3.2 Clases y librerías 23
3.3 API Telegram. 25
3.4 API de GPT-3 26
3.4.1 Parámetros 26
3.5 API de Twitter 28
Capítulo 4 Funcionalidades y características 31
4.1 Explicando el *few-shot learning*. 31
4.2 Creación fichero train.jsonl. 33
4.3 Funcionamiento general 35
4.4 Resultados 38
Capítulo 5 Conclusión y Líneas futuras 40
5.1 Conclusion 40

Capítulo 6 Bibliografía 42

Índice de figuras

Figura 1.1: Posición de los Transformers en el campo de la IA 11
Figura 2.1: Esquema sintetizado de un modelo embedding. [17] 12
Figura 2.2: Representación de Word embeddings en dos dimensiones. [18] 13
Figura 2.3: Esquema Transformers. [19] 14
Figura 2.4: Función solicitud completion hacia GPT-3 18
Figura 2.5: Formato fichero para búsquedas y preguntas/respuestas. 19
Figura 2.6: Formato fichero para clasificación. 19
Figura 2.7: Función para carga del fichero General 19
Figura 2.8: Formato fichero para Fine-tuning 21
Figura 2.9: Función carga de fichero para Fine-tuning 21
Figura 2.10: Método para Fine-tuning. 22
Figura 3.1: Diagrama UML de clases. 25
Figura 3.2: Configuración actual obtenida con el comando /status. 28
Figura 4.1: Selección de cabecera desde la interfaz con /config 32
Figura 4.2: Selección de frases de contexto predeterminadas. 33
Figura 4.3: Menú para reentrenamiento GPT-3 33
Figura 4.4: Conexión a API 2.0 Twitter. 34
Figura 4.5: Consulta base para la obtención de tweets. 34
Figura 4.6: Obtención de un hilo completo si el tweet es una respuesta.. 35

Índice de tablas

Tabla 1: Características de modelos y precios. 20
Tabla 2: Claves, tipos de API y protocolos de autenticación API Twitter ... 29
Tabla 3: Listado de comandos obtenidos con /help 38

Capítulo 1
Introducción

El procesamiento del lenguaje natural (NLP por sus siglas en inglés) es una disciplina de la inteligencia artificial destinada al estudio y tratamiento del lenguaje humano de forma computacional.

Hasta los años 90, básicamente los algoritmos funcionaban como árboles sintácticos basados en reglas gramaticales, luego, se introdujo el método estadístico, que fue una auténtica revolución, sobre todo en traducción automática. Este método consiste en trabajar con un corpus, y valorar en base al número de coincidencias por cada segmento, ya sea, palabras, frases, letras o incluso documentos enteros. Pero no fue hasta hace apenas 9 años, en 2013, cuando el NLP sufre un verdadero punto de inflexión, gracias a la aplicación de algoritmos de inteligencia artificial basados en redes neuronales, más concretamente algoritmos para redes neuronales profundas o Deep Learning.

Hay dos grandes algoritmos que han permitido un avance exponencial en el mundo del NLP, que son el *embeddings* de palabras, sobre todo con el algoritmo de Google Word2Vec inventado en 2013 y los Transformers, también creados por Google en 2017, algoritmos que se explicarán más adelante.

1.1 Motivación

Existía especial interés en crear un asistente virtual sobre incidencias informáticas, para poner a disposición de los usuarios. Los asistentes virtuales tipo *chatbots*, existen ya hace tiempo, pero la mayoría de ellos siguen siendo los denominados *chatbots* basados en reglas o recuperación, en los cuales, dada una entrada, una pregunta por parte del usuario, el sistema devolverá la respuesta más adecuada obteniéndola de una base de datos previamente preparada y adaptada a cada negocio. Estos chats son muy precisos en las respuestas, pero muy difíciles de mantener y actualizar, así como muy poco “inteligentes”, ya que funcionan de forma muy estática.

Pues bien, este TFG, crea un *chatbot* que funciona dentro del campo del Machine Learning, no tiene nada que ver con uno del tipo basado en reglas, lo que, hoy en día, puede parecer más inteligente, pero también, puede responder con imprecisión e incluso de forma incoherente, sobre todo semánticamente.

Además, la mayoría de *chats*, están optimizados para recibir preguntas en inglés, por lo que en este trabajo se trata de paliar esta carencia mediante la traducción automática y el entrenamiento.

1.2 Objetivos

Inicialmente, la idea era la creación de un *chatbot* sobre incidencias informáticas utilizando técnicas de NLP basadas en un modelo secuencial de redes neuronales recurrentes o sus variantes, entrenado y funcional, pero la irrupción en el mercado de modelos ya preparados con millones de datos y miles de millones de parámetros, cuyo entrenamiento es solo asumible por grandes compañías, como Google, Facebook o Microsoft, hizo que finalmente este trabajo se base en probar un modelo ya existente, en este caso el GPT-3 de OpenAI.

Así, que los objetivos conseguidos son, por una parte, la de crear una interfaz que sirva para analizar, comprobar, configurar, entrenar y usar GPT-3 desde cualquier dispositivo y por otro, crear un *chatbot* especializado inicialmente en respuestas sobre incidencias informáticas, pudiendo mantener el diálogo tanto en inglés como en español.

1.3 Estado del arte

La inteligencia artificial es un campo muy amplio, y lleno de términos y especificaciones, que a veces son confusas, sobre todo si no se tiene claro en donde estamos. Ya sabemos que NLP es la disciplina que computacionalmente se utiliza para el entendimiento y tratamiento del lenguaje humano. Pues bien, dentro de NLP los algoritmos de Deep Learning más comúnmente utilizados son los secuenciales, basados en redes neuronales recurrentes RNN, incluidas sus variantes como LSTM o GRU. El uso de modelos secuenciales viene de la necesidad de poder tener memoria, lo que en el campo del NLP es lo que hace referencia al contexto de una palabra u oración, de forma que la red neuronal no basará su salida solo en la entrada, sino también en entradas anteriores e incluso posteriores. El problema de estos algoritmos es que, al ser entrenados de forma secuencial, son demasiado costosos en tiempo, ya que no se puede aprovechar la potencia de procesamiento paralela que puede ofrecer, por ejemplo, cualquier GPU. Hay que decir, que también se pueden utilizar otros algoritmos de redes neuronales que pueden ser entrenados de forma paralela, como, por ejemplo, las redes neuronales convolucionales, aunque todas estas líneas de investigación han quedado eclipsadas con la aparición en 2017 de los Transformers, utilizados actualmente por los modelos del lenguaje más importantes, como el BlenderBot de Facebook, y el Bert de Google entre otros.

Dentro este enorme campo de la inteligencia artificial, los Transformers forman también el modelo base de GPT-3, que es la base de este *chatbot*. Pues tal y como se muestra en el diagrama, podemos decir son un tipo de algoritmo de modelo secuencial apoyado en redes neuronales profundas, que a su vez son realmente una extensión de las redes neuronales. Estas redes neuronales son un tipo de algoritmo de Machine Learning, y este son un tipo de algoritmos de la inteligencia artificial en general.

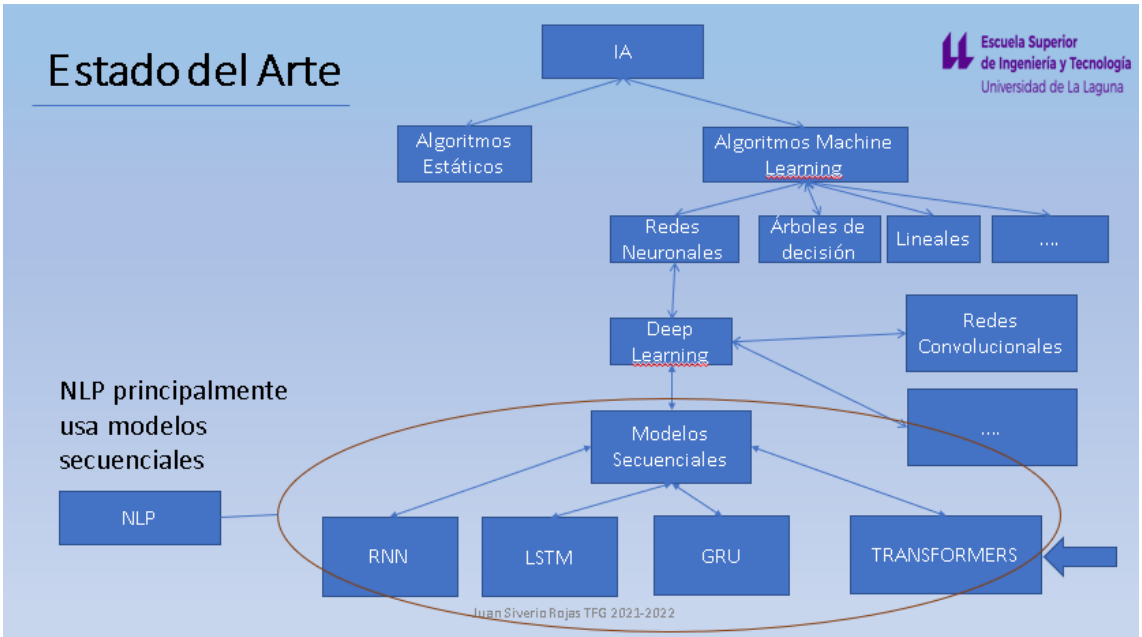

Figura 1.1: Posición de los Transformers en el campo de la IA

1.4 Estructura del documento

Aun intentando no centrarnos en explicaciones técnicas y excesivos conocimientos sobre Machine Learning, en el capítulo 2, se intentará dar una visión de en qué parte de este mundo de la inteligencia artificial se sitúan los Transformers, que son la base de la forma de funcionar de GPT-3, así como conocimientos teóricos básicos sobre características propias de los modelos del lenguaje incluidos en GPT-3. Tras estas explicaciones teóricas, el capítulo 3, se centrará en hablar de la programación en Python del *chatbot*, de las API, librerías utilizadas, así como de sus características.

A continuación, en el capítulo 4 se mostrará el funcionamiento del chat aportando ejemplos. Finalmente llegarán el apartado de conclusiones sobre las posibles mejoras de la aplicación y el rendimiento mostrado por los modelos GPT-3.

Capítulo 2
Fundamentos Teóricos

2.1 NLP: Embeddings y Transformers

Tal y como se adelantó en la introducción de este trabajo, desde mi punto de vista los dos hitos más revolucionarios en el procesamiento natural del lenguaje o NLP, son dos, los Word Embeddings y las Redes Transformers.

2.1.1 Embeddings

En el mundo del NLP, para poder analizar el lenguaje natural computacionalmente necesitamos transformar el texto en valores matemáticos que representen a una parte indivisible del texto en concreto.

Primero se usa la técnica de la *tokenización, que* consiste simplemente en dividir el texto en *tokens*. Cada *token único* puede ser comúnmente una letra o palabra, aunque también podría ser una oración completa e incluso un documento. Antes de obtener el *token* definitivo, el que será representado con un valor matemático único, se pueden efectuar infinidad de técnicas posibles y previas, como la derivación, lematización.

Una vez tenemos construido un vocabulario de *tokens* únicos extraídos del texto, es cuando a cada uno hay que asignarle un valor numérico. Aquí es donde entran en juego diferentes modelos matemáticos, que podría ser tan simples como enumerar cada *token* con un número consecutivo, mediante un vector *one-hot* o bien la forma de codificación más potente hoy en día, que son los ***Embeddings,*** consistentes en la representación de cada token mediante un vector denso, que indicará no solo a que tipo de token hace referencia, sino que también aportará algo de información extra dependiendo de cómo se haya obtenido ese vector denso.

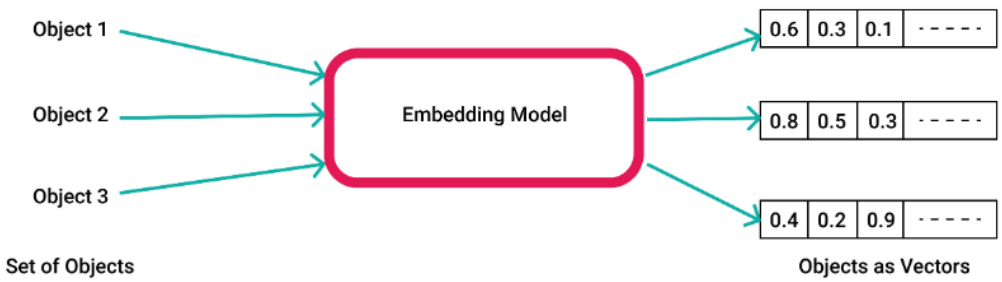

Figura 2.1: Esquema sintetizado de un modelo embedding. [17]

2.1.2 Word Embeddings

Sería realmente deseable que el número que identifica a cada *token,* aporte además información extra como, por ejemplo, tipo de tokens, contexto, similitud. Pues utilizando los *embeddings* (esos vectores densos), y centrándonos en el caso de que un *token* represente a una palabra, nacen diferentes algoritmos, a los que llamaremos *Word Embeddings*, que vendría a significar algo así como encaje de palabras, porque hacen el encaje matemático de un espacio con una dimensión por palabra a un espacio vectorial continuo con menos dimensiones, donde el primer espacio es representado.

Estos algoritmos básicamente funcionan extrayendo determinada información del texto completo, como, por ejemplo: número de veces que se repite la palabra, distribución de probabilidades de cada palabra vecina o repeticiones, entre muchas otras. Pues bien, toda esta información queda codificada en el vector denso *embeddings* de la dimensión elegida.

Dentro de este tipo de algoritmos, uno de los modelos más potentes es el *Word2Vec*, creado por Tomas Mikolov en Google en 2013 y que está basado en redes neuronales para la obtención de los vectores densos. Este trabajo incluyó dos modelos diferentes de aprendizaje, el *CBOW* para predecir dada una palabra, un contexto y el *Skip-Gram*, que dado una palabra nos devuelva un posible contexto. Los *embeddings* creados con estos modelos, tienen la característica de cercanía, o sea, que los vectores que representan dos palabras semánticamente similares serán cercanos en el espacio del *embedding*. Matemáticamente, la distancia entre dos vectores del *embedding* se mide con la distancia llamada similitud de coseno, que consiste simplemente en el coseno del ángulo que forman esos vectores. Esta distancia no depende de las magnitudes de los vectores y se puede hallar de forma muy sencilla mediante la expresión del producto escalar.

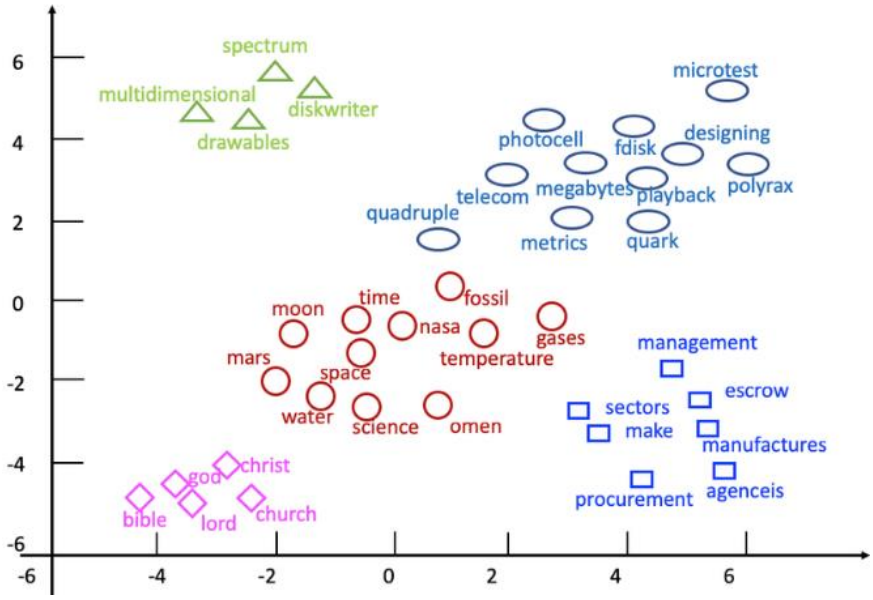

Figura 2.2: Representación de Word embeddings en dos dimensiones. [18]

2.1.3 Transformers

Como ya se ha avanzado, los modelos secuenciales, son aquellos en los que ni los *inputs* ni los *outputs*, tienen por qué tener un tamaño fijo, al contrario que en el resto de los algoritmos basados en redes neuronales. Estos modelos, tienen la capacidad de relacionar diferentes entradas en el tiempo devolviendo una salida en consecuencia y dando una apariencia de *memoria*. De esta forma, solucionan los problemas de contexto (información previa y/o posterior) que es muy importante a la hora de predecir. Estos modelos secuenciales son las redes neuronales recurrentes o RNN. Tras estos modelos llegaron los GRU y LSTM, que incluyeron algunas mejoras sobre todo en la cantidad de contexto a recordar.

Pues bien, los Transformers cuya propuesta inicial fue presentada en un *paper* creado por desarrolladores de Google en 2017, al que llamaron *"Attention Is All You Need"* son la evolución de estos modelos secuenciales, cuya ventaja, entre otras, es que disponen de una memoria del contexto superior a los modelos secuenciales descritos anteriormente. Además, tienen otro punto que las distingue de sus antecesoras y es que pueden ser entrenadas de forma paralela, de ahí que GPT-3 haya podido ser entrenado con una cantidad enorme de datos provenientes de internet, y conteniendo más de 175,000 Millones de parámetros.

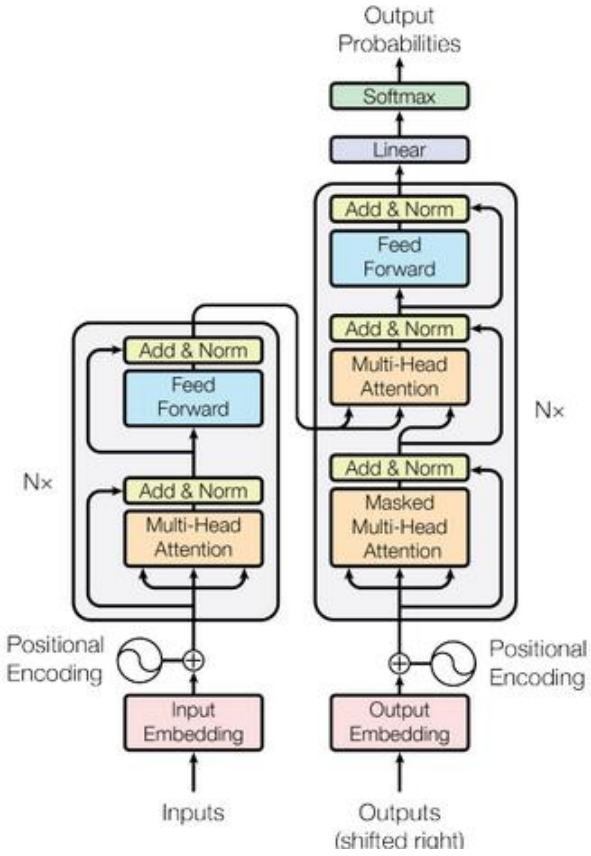

Figura 2.3: Esquema Transformers. [19]

La idea clave del Transformer es la de gestionar completamente las dependencias entre la entrada y salida sin necesidad de la recurrencia. Al mecanismo que permite al modelo saber con qué otra palabra de la secuencia está relacionada la palabra que se procesa en ese instante de tiempo, se le llama autoatención. En los modelos secuenciales anteriores, es necesario presentar las palabras (tokens) en el natural en el que aparecen a la entrada del modelo. Con los Transformers esto no es necesario y se pueden tratar todas las palabras de forma simultánea, dando ese paralelismo necesario para agilizar el entrenamiento. La forma de conocer la posición de cada palabra es gracias el mecanismo de posicionamiento, sustituyendo así a la recurrencia.

Como se puede ver en la figura 2.3, se pueden diferenciar fácilmente los dos componentes fundamentales, el *Encoder* (bloque parte izquierda) y el *Decoder*(bloque parte derecha). Para que esto funcione, se ha de trabajar con *embeddings*.

Básicamente y sin entrar en detalles, se comprueba la similitud de coseno de cada palabra con la otra.

2.2 GPT-3

GPT-3 es un modelo del lenguaje, que fue presentado oficialmente el 28 de mayo de 2020 por la compañía OpenAI, impulsada por Elon Musk y actualmente asociada con Microsoft. GPT-3 dispone de varios modelos distintos, todos ellos basados en Transformers.

OpenAI, liberó la API de esta tecnología el 18 de noviembre de 2021, o sea, hace muy poquito, permitiendo a desarrolladores poder utilizarlo, realizar pruebas, e incluso implementarlo en sus aplicaciones. Y aunque eso no quiere decir que sea gratuito, inicialmente se dispone de un pequeño saldo a favor para poder utilizarlo.

La comunidad ha creado librerías para prácticamente cualquier lenguaje de programación, aunque las API oficiales están diseñadas solo para API RESTFull, Python y Node.js. Esto garantiza la portabilidad y disponibilidad de este motor en prácticamente cualquier plataforma existente.

OpenAI, dispone de una interfaz, previa autenticación, para poder juguetear y probar el modelo, personalizando diferentes valores. Esta interfaz web, es lo que llaman el *PlayGround*.

2.1.1 Glosario de términos

Hay algunos términos relacionados con GPT-3 que deberíamos de conocer para comprender mejor el resto de este texto. La mayoría de estos términos se explican con más detalle en apartados posteriores, así que no se preocupe si no entiende el significado de alguno de ellos, simplemente tome este glosario como una referencia rápida.

Prompt: Es el input para GPT-3, suele ser una pregunta, comentario, anotación o, en definitiva, cualquier texto escrito para ser enviado por parte del usuario.

Completion: Es una cadena de texto que corresponde a la contestación del modelo al *prompt* recibido. Puede ser una respuesta, completar un texto, una traducción o incluso la respuesta a una orden.

Tokens: Un token puede ser una letra, una palabra, una oración e incluso un documento completo, pero en el caso de todos los modelos de GPT-3, los tokens equivalen aproximadamente a algo menos de una palabra, con lo que, 1000 tokens, serían unas 750 palabras en inglés. Es la base para la monetización por parte de OpenAI.

Modelos Base GPT-3: También llamados modelos antiguos ya que han sido los recomendados hasta el año 2021. Estos modelos son el *davinci, curie, Babbage y ada*. Con estos modelos, a parte de las tareas de *completion(terminación)*, se pueden realizar tareas de clasificación, búsqueda, preguntas/respuestas o *Fine-tuning*.

Modelos Instruct: Son los modelos nuevos y recomendados desde el año 2021. Están optimizados para responder mejor siguiendo instrucciones. Estos modelos son el *text-davinci-002, text-davinci-001, text-curie-001, text-babbage-001 y text-ada-001*. Solo realizan de forma nativa tareas de *completion* no permitiendo ser ajustados con *Fine-tuning*. Las tareas de clasificación, búsqueda y preguntas / respuestas pueden simularse con técnicas de *few-shot learning*.

Modelos Codex: Son un conjunto de modelos que están optimizados para entender y generar código en varios lenguajes de programación, incluyendo la traducción del lenguaje natural a código.

Modelos Content-Filter: Es un modelo personalizado que permite detectar texto que puede ser sensible o inseguro.

Few-shot learning: Forma de aprendizaje de los modelos, que se basa en diferentes técnicas para formar un ***prompt*** adecuado dependiendo de la tarea que queramos realizar. Podemos diseñar ***prompts*** adecuados dependiendo de si lo que se desea son para tareas de clasificación, preguntas/respuestas, traducción, generación de texto y mucho más. (Se explicará más en detalle en el apartado 4.1)

Fine-tuning: Capacidad que tienen solo los modelos base GPT-3 para ser

reentrenados, permitiendo así disponer de modelos personalizados. (Se explicará más en detalle en el apartado 2.4.5)

***PlayGround*:** Aplicación web de OpenAi, habilitada tras autenticación previa para probar el funcionamiento de los modelos GPT-3. Esta web, permite la modificación de motores, parámetros de funcionamiento y algunos otros valores para probar de forma rápida diferentes funcionalidades. Es una ayuda muy interesante para el desarrollador.

2.1.2 Modos de funcionamiento

GPT-3 ha sido entrenado con miles de millones de datos, mayoritariamente en inglés y optimizado para la generación de texto, esto es, dada una entrada, será capaz de continuar escribiendo, completando la entrada, de forma coherente como si de un humano se tratara. Este modo de trabajar es el llamado ***completion (algo así como terminación),*** y es el más utilizado.

2.1.2.1 Modelos Instruct

En estos modelos, el único modo de funcionamiento es el ***completion***. No obstante, la gran cantidad de información con la que han sido entrenados estos modelos, han dado un resultado tan impresionante, que incluso con solo utilizar la técnica del ***few-shot learning***, se puede conseguir modos de funcionamiento del tipo clasificación, preguntas/respuestas o búsquedas, sin necesidad de haber sido exclusivamente entrenados para ello. Se expondrá un ejemplo práctico de esta técnica aplicado al funcionamiento en modo chat en el apartado 4.1.

Un ejemplo en Python, de cómo realizar una solicitud en modo ***completion*** sobre cualquier cosa, tendría el aspecto de la figura 2.4. Los parámetros del método, se explicarán en profundidad en el apartado 3.4.1.

```
import  openai

openai.Completion.create(
prompt = "Háblame de Don Quijote de la Mancha"
engine="text-davinci-002",
temperature= 0,5,
top_p = 1,
max_tokens = 512,
…
)
```

Figura 2.4: Función solicitud completion hacia GPT-3

2.1.2.2 Modelos Base GPT-3

Estos modelos no solamente pueden ser utilizados en modo ***completion*** al igual que los *instruct*, además se pueden utilizar para mejorar la funcionalidad en los modos *clasificación(classifications), búsquedas(searches) y preguntas/respuestas (Answers)*, se pueden adaptar con una base de datos propia y optimizar para cada uno de estos modos de funcionamiento en concreto.

OpenAI está implementando de forma nativa estas características en estos modelos a través de su API, y aunque aún en versión beta, son perfectamente funcionales, teniendo a disposición de los desarrolladores métodos específicos para clasificación, búsqueda y preguntas/respuestas.

Por si fuera poco, estos modelos también permiten el *Fine-tuning*, que viene a significar que se puede realizar un ajuste fino del modelo, o sea, es una forma de reentrenamiento con información personalizada.

2.1.2.3 Carga de Ficheros

Los métodos *classifications, searches, Answers y Fine-tuning* solo funcionan con los motores base de GPT-3, también llamados "antiguos" y trabajan cargando la información contenida en ficheros no mayores de 150 Mb cada uno, pudiendo sumar entre todos un máximo de 1GB. Por lo tanto, primero se ha de realizar la carga del fichero de información. Todos estos ficheros deben estar formato ***JSON Lines (.jsonl),*** el cual se compone de una entrada en formato ***Json*** por cada línea. Cada línea tendrá un formato distinto dependiendo de la tarea para la que se requiera.

A continuación, sin entrar en demasiado detalle se muestran los distintos formatos de fichero para clasificación, búsquedas y preguntas/respuestas. La funcionalidad de *Fine-tuning* se verá más adelante y con más detalle en el apartado

2.4.5.

Búsquedas *(Search)* y Preguntas/Respuestas *(Answers)* utilizan el mismo formato de fichero. El campo *"metadata"* es opcional:

```
Search-answers.jsonl

{"text": "texto1", "metadata": "texto descriptivo del campo text"}
{"text": "texto2", "metadata": "texto descriptivo del campo text "}
```

Figura 2.5: Formato fichero para búsquedas y preguntas/respuestas.

Clasificación *(Classifications)* usa este formato. El campo *"metadata"* es opcional:

```
classifications.jsonl

{"text": "texto1", "label": "Positivo", "metadata": "{"origen":"ejemplo"}}
{"text": "texto2", "label": "Negativo", "metadata": "{"origen":"ejemplo"}}
```

Figura 2.6: Formato fichero para clasificación.

La carga de estos ficheros se realiza mediante la función que aparece en la figura 2.7 y en la que es importante prestar atención a la variable *"purpose"* puesto que es la encargada de especificar el propósito del fichero y que en base esto la API pueda validar que tenga el formato correcto.

```
import openai

openai.File.create(
file = open(fichero .jsonl),
purpose="Search" / purpose = "Classifications" / purpose = "Answers"
)
```

Figura 2.7: Función para carga del fichero General

2.1.3 Límite Prompts + Completions

Dependiendo del modelo, se podrá usar una combinación de ***prompt + completion*** con un tamaño de tokens máximo establecido. Esto quiere decir que no se puede exceder ese límite en ninguna solicitud, por lo que hay que tener en cuenta que la suma de tokens totales es equivalente a la suma de los tokens contenidos en los ***"completion"*** más los contenidos en los ***"prompt",*** siendo estos últimos los más

costosos, ya que normalmente el input para GPT-3 no será solamente la última frase escrita por parte del usuario, sino que además contendrá algunas frases anteriores, a modo de memoria. Estas frases incluirán normalmente los ***prompts*** y los ***completions*** anteriores, sin superar nunca el límite de tokens del modelo y servirán como un contexto que ayudará al motor a responder de forma más coherente y lo más relacionado posible a lo que se está hablando. Actualmente los límites de tokens de los modelos están establecidos en 2.048 o 4.000.

2.1.4 Monetización

GPT-3 no es gratuito, la forma de monetizar por parte de OpenAI se basa en el consumo de cada 1000 tokens vía solicitud. Pues bien, dependiendo del modelo a utilizar, se aplicará un precio u otro, siendo los modelos más potentes bastante más caros con respecto a modelos inferiores. También variará el precio dependiendo de si es un modelo ajustado con *Fine-tuning*.

En la tabla 1, se muestra la categorización de algunos modelos que engloba varios de los conceptos explicados:

Motor	**Descripción**	**Max Tokens**	**Base de datos Actualizada hasta**	**Precio ($) Uso / 1K tokens**	**Precio ($) Entrenamiento modelo *Fine-tuning* / 1k tokens**	**Precio ($) Uso modelo *Fine_tuning* / 1k tokens**
text-davinci-002	*Modelos Instruct*	*4.000*	*Junio 2021*	*0.0600*	*No se puede Entrenar*	
text-curie-001	*Modelos Instruct*	*2.048*	*Octubre 2019*	*0.0060*	*No se puede Entrenar*	
text-babbage-001	*Modelos Instruct*	*2.048*	*Octubre 2019*	*0.0012*	*No se puede Entrenar*	
text-ada-001	*Modelos Instruct*	*2.048*	*Octubre 2019*	*0.0008*	*No se puede Entrenar*	
Davinci	*Modelo base GPT-3*	*4.000*	*Junio 2021*	*0.0600*	*0.0300*	*0.1200*
Curie	*Modelo base GPT-3*	*2.048*	*Octubre 2019*	*0.0060*	*0.0030*	*0.0120*
Babbage	*Modelo base GPT-3*	*2.048*	*Octubre 2019*	*0.0012*	*0.0006*	*0.0024*
Ada	*Modelo base GPT-3*	*2.048*	*Octubre 2019*	*0.0008*	*0.0004*	*0.0016*

Tabla 1: Características de modelos y precios.

2.1.5 Fine-tuning

Gpt-3, tal y como se indica en el glosario de términos, permite lo que llaman *Fine-tuning*, que no es más que la capacidad de personalizar un modelo ya existente mediante el entrenamiento. Recordemos que los modelos de GPT-3 han sido

entrenados con millones de datos de cualquier índole, pero con esta funcionalidad, tenemos la posibilidad de poder especializar nuestro modelo en las cuestiones que nos interesen. Como ya se ha dicho los modelos que GPT-3 permite que sean reentrenados, no son todos, de hecho, solo a los modelos base o modelos viejos, se les puede realizar el *Fine-tuning*.

Otra gran ventaja de reentrenar el modelo es la de ahorrar costes, ya que la monetización se realiza en base al número de tokens consumidos y en un modelo sin ajustar, la única posibilidad de obtener una respuesta lo más precisa posible, es usando la técnica del *few-shot learning*, la cual está basada básicamente en realizar una solicitud con un *prompt* con algo de contexto previo, lo que provoca un alto consumo de tokens. Por el contrario, con un modelo *Fine-tuning*, una vez entrenado, se puede conseguir la respuesta deseada simplemente con una simple entrada. El resultado es que normalmente se ahorra dinero incluso aunque los precios por cada *token* utilizando un modelo ajustado sea el doble que el de un modelo sin ajustar.

Para ajustar el modelo, se necesita la carga de ficheros tal y como se explicó en el apartado 2.4.2.2.1, pero en este caso el formato del fichero se compone de un prefijo llamado ***"prompt:"*** para los *inputs*, que para la utilidad de esta aplicación equivalen a preguntas, y seguido del prefijo ***"completion:"*** más él texto que hará de respuesta. También se agregan al final de cada campo caracteres de *stop*, lo que es una buena práctica para ayudar tanto al modelo como a la aplicación a saber cuándo finaliza una pregunta o respuesta. Así, que el formato del fichero sería algo parecido al de la figura 2.8.

```
{"prompt": "texto<caracteres stop>", "completion": "Texto generado<caracteres stop>"}
{"prompt": "texto2<caracteres stop>", "completion": "Texto generado<caracteres stop>"}
```

Figura 2.8: Formato fichero para Fine-tuning

En Python la carga de ficheros con propósito de *Fine-tuning* se realiza con el método de la figura 2.9

```
import opeai

openai.File.create(
file = open( fichero .jsonl),
purpose = "fine-tune"
)
```

Figura 2.9: Función carga de fichero para Fine-tuning

El método incluido en la clase *Gpt3()* para importar el fichero de entrenamiento y aplicar el *Fine-tuning* es el que se muestra en la figura 2.10. La creación del fichero

train.jsonl se explica más adelante en el apartado 4.2.

```
def my_ft_model(self, file_for_train="train.jsonl"):
                self.info_file= openai.File.create(file=open(file_for_train),purpose="fine-tune")
                self.id_file=self.info_file["id"]
                self.info_model_ft= openai.FineTune.create(training_file=self.id_file)
```

Figura 2.10: Método para Fine-tuning.

Capítulo 3
Implementación

3.1 Aplicación

En un chat, una de las características que debe de valorarse, es la posibilidad de acceder al mismo desde el mayor número de plataformas posible. En ese sentido, la aplicación denominada Telegram, tiene soporte para prácticamente todos los sistemas operativos, Windows, MacOS, Linux, Android, IOS e incluso acceso vía web. Además, es una aplicación con más de 500 millones de usuarios activos actualmente y, por si fuera poco, permite la implementación de *bots*. Todo ello de forma completamente gratuita.

Por eso, *la interfaz para el usuario será la propia aplicación de Telegram*, disponiendo así de un software ya creado, multiplataforma y con un diseño pensado para funcionar como un *chat.*

Un *bot*, simplemente es una cuenta virtual que no está asociada a ningún usuario. El ***bot*** creado se llama **RMPixel_bot**, y es completamente público, de forma que puede ser agregado como contacto por cualquier usuario existente de Telegram, incluso desde grupos o canales.

Este Bot es administrado por una aplicación que correrá en cualquier equipo, a modo de servidor y será nuestro asistente virtual. La aplicación que está programada en su totalidad en el lenguaje de programación Python 3.6, será la base de este TFG.

Mediante la interfaz, esta aplicación es capaz de recibir comandos, distinguiéndose por comenzar con una barra invertida "/". Se han definido comandos para permitir la configuración de todos los parámetros que permite *PlayGround* y además, se han definido muchos otros que admite la API y que no son configurables desde la web. También se han agregado comandos para las funcionalidades extras que permite la aplicación, como la traducción, definir encabezados y contexto o configuración del entrenamiento desde cuentas de Twitter.

3.2 Clases y librerías

Se han programado tres clases:

Gpt3: Esta clase, se encuentra en el fichero *gpt3.py*, es la encargada de la configuración y administración de los modelos GPT-3. Entre sus capacidades está la de modificación de parámetros, cargar ficheros para entrenamiento, envío de preguntas,

recepción de respuestas y mucho más.

Twitter: Esta clase, se encuentra en el fichero *twitter.py,* es la encargada de la conexión a Twitter, desde la que se obtendrá los ***tweets*** de los últimos 7 días relacionados con determinadas cuentas. También se encarga de formatear los ***tweets*** recopilados para crear el fichero ***.jsonl*** necesario para realizar el *Fine-tuning* con el motor seleccionado.

RMPixel_Bot: Esta clase, se encuentra en el fichero *RMpixel_Telegram.py,* y es la clase base y principal. Es la única que el desarrollador tendrá que conocer y programar, puesto que implementa todos los métodos necesarios para la creación y administración del *bot*. Con solo llamar al método *start_bot()*, comienza el funcionamiento de todo el *chat* con valores predeterminados. Esta clase se encarga de administrar la interfaz de Telegram, incluyendo recepción de mensajes o comandos por parte del usuario, conexión con GPT-3, conexión con *Google Translate* y conexión con Twitter. Internamente, hace uso de las clases Gpt3 y Twitter.

Además, se ha requerido el uso de las siguientes librerías:

- **Python-telegram-bot**: Telegram solo dispone para su administración de una API REST. En su lugar, se ha utilizado esta librería de la comunidad para la administración, control y conexión con los servidores de Telegram.

- **OpenAI**: Esta librería es la encargada de la conexión con los diferentes motores de GPT-3. Es proporcionada y mantenida de forma oficial por la compañía del mismo nombre, en sus ediciones para Python y Node.js. Aquí se ha utilizado la versión para Python.

- **GoogleTrans**: Esta librería permite realizar funciones de traducción utilizando los motores de *Google Translate*. Se utiliza cuando se activa la traducción automática en el chat, la cual se explicará en más abajo.

- **Tweepy**: Esta librería es la encargada del acceso y recuperación de mensajes de Twitter. Es una librería de la comunidad, completamente compatible con la nueva versión 2.0 de la API de esta plataforma y con el nuevo sistema de autenticación OAuth 2.0.

En la figura 3.1, se muestra el diagrama de clases con atributos y métodos simplificados.

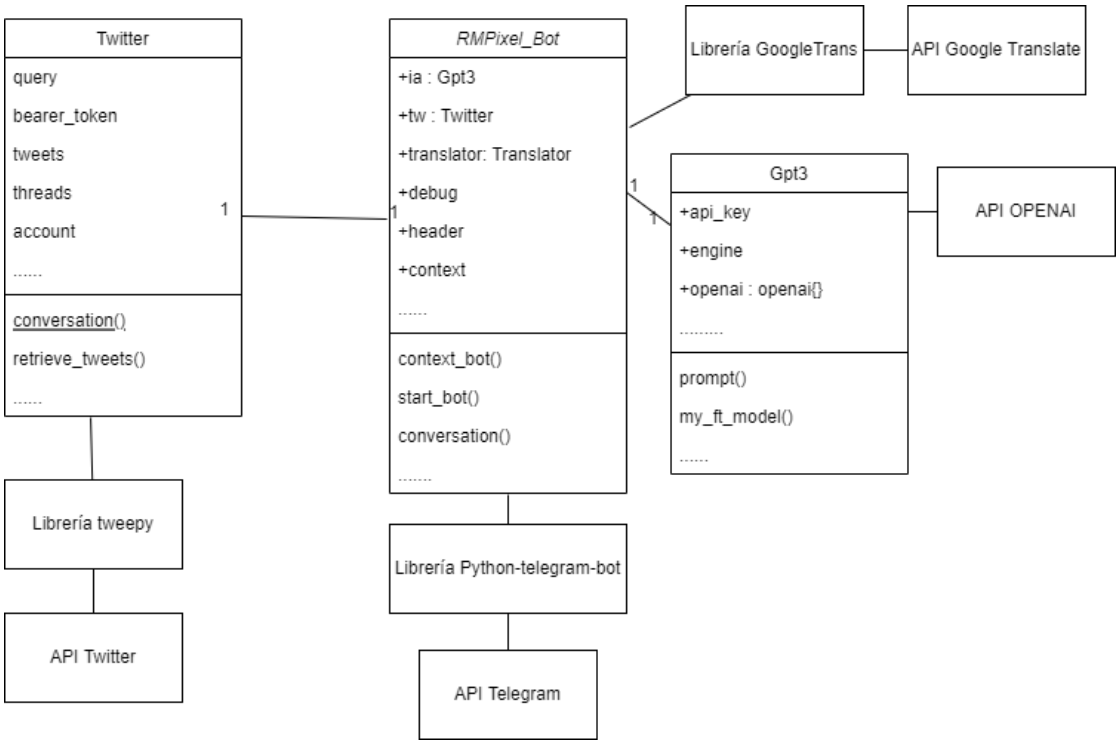

Figura 3.1: Diagrama UML de clases.

3.3 API Telegram.

Los servidores de Telegram, administrarán la encriptación y comunicaciones entre los usuarios y Telegram, haciendo de intermediarios entre el usuario y nuestra aplicación. Para acceder a la información de esos servidores, Telegram ofrece una API REST a través de HTTP, aunque próximamente solo las conexiones HTTPS serán las permitidas. Esta API dispone de dos modos de funcionamiento:

WebHook: En esto modo de funcionamiento, son los servidores de Telegram los que se ponen en contacto con la aplicación, enviándoles cada nuevo mensaje recibido en cuando disponen de él. Es un modo más rápido que el ***Polling***, que consume menos recursos, pero requiere de una instalación de la aplicación en un servidor con configuraciones más complejas, normalmente con IP públicas, redirección de puertos en router, dns, firewall.

Polling: En este modo la aplicación realiza consultas de forma permanente y continua a los servidores de Telegram para comprobar si hay nuevos mensajes. Esta forma puede ser más lenta y costosa, pero permite el funcionamiento tras la mayoría de firewalls y desde prácticamente cualquier equipo con conexión a internet.

La comunidad, ha programado varias librerías para prácticamente cualquier lenguaje de programación basadas en esta API REST, y en esta aplicación se utiliza

Python-telegram-bot ya que dispone de una documentación en línea bastante clara.

Toda la comunicación con los servidores de Telegram se realiza mediante un token único que nos permite identificarnos en nombre de un *bot* en concreto previamente creado. Este proceso es muy sencillo, ya que simplemente cualquier usuario dado de alta en Telegram, debe de buscar un contacto denominado "***BotFather***", que no es más que el administrador de *bots.*

Con el comando */help*, se obtiene un listado de las opciones de configuración. Todos los comandos en Telegram funcionan anteponiendo una barra invertida "/" delante. El que nos interesa es */newbot,* con el que crearemos un *bot*, al que se le pone un nombre y configuran diferentes opciones. Una vez creado, se dispone de un token único, que identificará inequívocamente ese *bot*. Todo el acceso a los métodos de la API se hace adjuntando ese token.

Esta API es administrada completamente por la clase **RMPixel_Bot,** y en este caso, el token obtenido identifica al *bot* al que se ha llamado igual que a la clase, o sea, **RMPixel_bot**.

3.4 API de GPT-3

Para utilizar GPT-3, el primer paso será darse de alta en https://openai.com/api. A fecha de hoy, se puede crear una cuenta gratuita con un saldo de regalo a favor, para realizar todo tipo de pruebas, de 18 dólares durante un máximo de 3 meses.

Hecho esto, se nos proporcionará una ***API key***, que funciona exactamente igual que el caso del ***token*** para la API de Telegram. En este caso, se usa la librería oficial de OpenAI para Python, llamada ***openai.***

Cualquier petición que se realiza a GPT-3, una vez entrenado, configurado o simplemente utilizando las opciones predefinidas, es realizada a través del método *prompt(self, text),* accedido automáticamente desde la clase **RMPixel_Bot.** Un resumen del proceso es que una vez descartado que no se ha recibido un comando, el texto introducido por el usuario es formateado dependiendo de la configuración, agregando contexto, cabecera y/o palabras de inicio, e incluso traducido, para luego pasarlo como parámetro al método *prompt()* encargado de realizar la solicitud a GPT-3 con la configuración establecida.

3.4.1 Parámetros

La aplicación utiliza inicialmente valores por defecto, que podrán ser modificados

con cualquier de los comandos disponibles desde la misma interfaz de Telegram. Los parámetros principales para modificar el comportamiento de GPT-3 son los mostrados a continuación.

Engine/model: *Valor por defecto "text-curie-001". Comando desde la interfaz: /config*

El identificador del motor a utilizar, pueden ser *motores base de GPT-3* o *motores nuevos instruct*. Para el caso de modelos *Fine-tuning* se utilizará la variable "*model"* y no "*engine",* en su lugar.

max_tokens: *Valor por defecto 512. Comando desde la interfaz: /max_tokens valor*

Valor entre 0 y 4000 según modelo. Número máximo de *tokens* devueltos en una respuesta. Recordar que el número de *tokens del prompt* más el *completion* no puede exceder el límite del modelo. La aplicación controla el tamaño máximo del *prompt*, impidiendo enviar uno que supere el límite de tokens del modelo menos el valor *max_tokens*.

Cuando se hace una solicitud al modelo, este internamente considerará varias respuestas posibles, basándose en la probabilidad o incluso en si ya ha respondido de forma similar recientemente. Podemos controlar cuantas respuestas obtener, e incluso aplicar algo de aleatoriedad modificando algunos parámetros. Con esta aplicación se permite modificar los siguientes:

n: *Valor por defecto 1. Comando desde la interfaz: /n valor*

Número de respuestas a obtener.

top_p: *Valor por defecto 1. Comando desde la interfaz: /top_p valor*

Valor entre 0 y 1 que indica el porcentaje de respuestas con mayor probabilidad sobre las que se aplicarán el resto de los parámetros. Por ejemplo, 0.2 significaría que se tendrán en cuenta el 20% de las respuestas posibles con mayor probabilidad.

temperature: *Valor por defecto 0. Comando desde la interfaz: /temperature valor*

Valor entre 0 y 1 de aleatoriedad. Sobre el número de respuestas obtenidas con el parámetro ***top_p***, se elegirá al azar tantas respuestas como indique el parámetro ***n.***

Por ejemplo, un valor de 0 indicará que no se aplicará ninguna aleatoriedad y por consiguiente se elegirá la respuesta disponible con mayor probabilidad. Esto es útil para obligar al modelo a devolver siempre la misma respuesta a la misma pregunta, y por tanto ser útil cuando se requieren respuestas bien definidas, por el contrario, si se requiere de un chat más creativo, habría que utilizar valores superiores a 0.

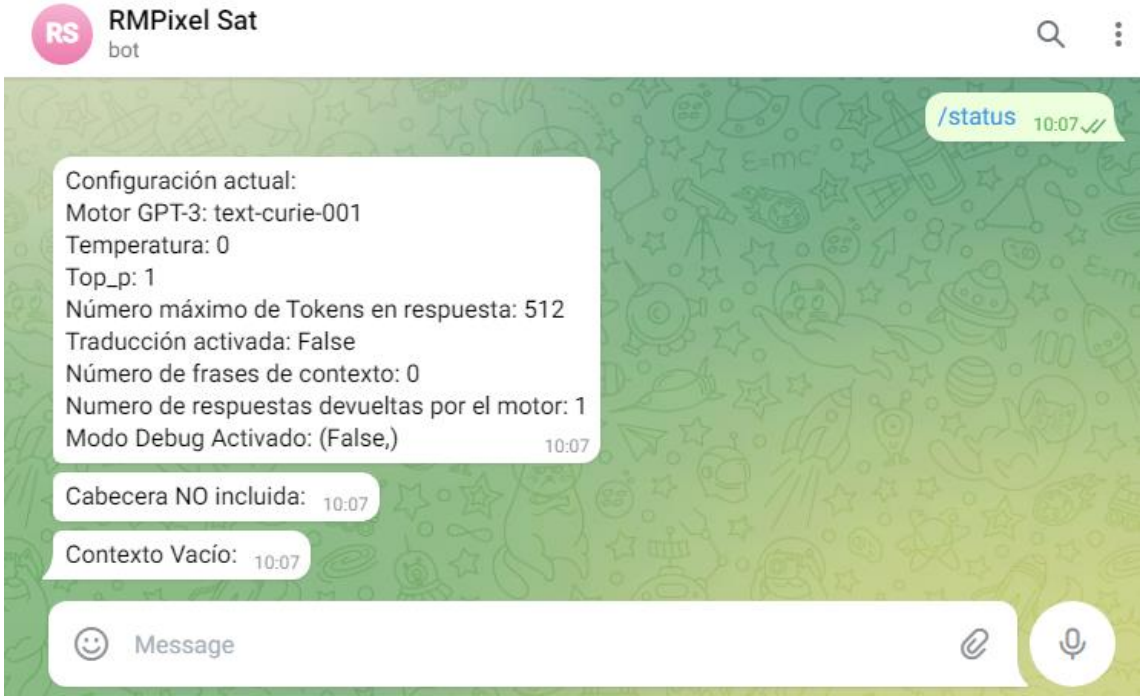

Figura 3.2: Configuración actual obtenida con el comando /status.

3.5 API de Twitter

Como base de conocimiento para reentrenar a los modelos GPT-3 para su especialización, mediante *Fine-tuning*, la aplicación utiliza la red social Twitter para la obtención de la nueva información.

Esta decisión está motivada por varios motivos.

1. El funcionamiento de Twitter basado *tweets*, (son pequeños textos con un máximo de 280 caracteres), es un tamaño ideal para la monetización basado en *tokens* de OpenAI. De esta forma, se ahorran costes al utilizar *"respuestas cortas"*
2. Gran variedad de especialización puesto que la información que se obtiene depende en gran medida de las cuentas de las que se recogen los *tweets*.

   Por ejemplo, para obtener tweets para el *chatbot* de este TFG, se ha usado la cuenta **@MicrosoftAyuda**, (cuenta oficial de Microsoft en español para atención al cliente) que usa un formato similar al que queremos utilizar con nuestro *chatbot*, o sea, un asistente de incidencias informáticas en español. Se usará la cuenta @MicrosoftHelps (cuenta oficial de Microsoft) para contenido principalmente en inglés.
3. El acceso a la API es sencillo, gratuito y con disponibilidad de librerías por parte de la comunidad para la mayoría de los lenguajes de programación. La red social tiene disponibles dos APIs:
   a. Streaming APIs: Esta API permite la descarga y monitorización de tweets en tiempo real.

b REST. Esta API permite el acceso a tweets con un atraso en el tiempo de alrededor de dos horas. También permite la administración de Twitter, e incluso actuar en nombre de una cuenta. Con esta API se pueden crear/descargar tweets, leer información del usuario, *followers* y mucho más. Es la utilizada en esta aplicación.

Los requisitos para poder acceder a la API de Twitter como desarrolladores son, por una parte, disponer de una cuenta de esta red social y después darla de alta como desarrollador en la plataforma habilitada para ello a través del portal https://developer.twitter.com.

Ahí deberemos de crear una App y entonces se nos facilitarán cinco *keys* que permitirán diferentes tipos de autenticación para poder utilizar la API.

Se dispone de autenticación como aplicación (lo cual permite el uso de la API para la lectura de tweets) o en nombre de una cuenta Twitter (que permite realizar todas las operaciones que podría hacer cualquier usuario a través de la interfaz), todo ello mediante los protocolos *"OAuth 1.0a User Context" o "OAuth 2.0 Bearer Token".*

En la tabla 2, se muestra una simplificación de las posibles combinaciones de protocolos y *keys.* Hay que destacar que el *bearer_token* solo puede ser utilizado de forma independiente.

			REST API
	REST API	***REST API***	***Streaming API***
Keys	**Autenticación de aplicación: OAuth 1.0a**	**Autenticación de aplicación: OAuth 2.0**	**Autenticación como usuario: OAuth 1.0a**
Consumer_key	X	X	X
Consumer_secret	X	X	X
Access_token			X
Access_token_secret			X
Bearer_token		X	

Tabla 2: Claves, tipos de API y protocolos de autenticación API Twitter

Como lo que se pretende es recuperar *tweets* públicos realizando operaciones de solo lectura, se usa la autenticación basada en *Bearer_token*, o sea, autenticación basada en aplicación utilizando el protocolo OAuth 2.0. Además, se usa la última versión de la API, la 2.0, liberada en 2020 y que está programada desde cero con total compatibilidad de todas sus funciones con este nuevo protocolo.

Todos los métodos de acceso a la API tienen limitaciones o restricciones en el número de solicitudes que se pueden realizar en un tiempo determinado, que pueden variar dependiendo de la versión de API a utilizar, del protocolo de autenticación, del tipo de API (REST o Streaming), de la suscripción y del método en concreto a utilizar. En este caso, utilizando la versión 2.0 de la API, con autenticación de aplicación mediante

OAuth 2.0, la suscripción es la predeterminada llamada *Essentials* y con el método *recent search* (para búsqueda de tweets recientes de los últimos 7 días) se tiene un *rate limit* de 450 solicitudes cada 15 minutos, lo que teniendo en cuenta que el máximo número de *tweets* para este método es de 100 por cada petición, resulta en la obtención de 45.000 *tweets* cada 15 minutos, no superando el límite mensual de esta suscripción que está establecido en los 500.000 *tweets*.

Se puede elevar la suscripción de desarrollador al tipo *Academic Research* para conseguir acceso al método *full-archive search,* que permite recuperar 500 *tweets* por petición con un máximo de 10 millones al mes y romper la barrera de los últimos 7 días, obteniendo *tweets* desde los inicios, en Marzo de 2006.

Capítulo 4
Funcionalidades y características

Una vez explicada gran parte de la teoría y características que subyacen detrás de la aplicación, términos, APIs y parámetros, en los siguientes apartados de este capítulo se hace un resumen del funcionamiento de algunos puntos fuertes de la aplicación.

4.1 Explicando el *few-shot learning*.

GPT-3 nos responderá de diferente manera dependiendo en gran medida del contexto, esto es, las frases anteriores. La forma de preparar este contexto previo es lo que se conoce como el *few-shot learning* descrito en el apartado 2.4.1. Podemos separar este contexto en dos partes, *la cabecera y las frases.*

La cabecera es una breve descripción que hará de guía al motor, así como pequeñas órdenes de cómo queremos que responda, que lenguaje debe de utilizar y otros matices que debe de conocer el chat. En este chat podemos seleccionar una cabecera en inglés o en español y como la finalidad es un chat para asistencia informática, y deseando que GPT-3 crea que es una inteligencia artificial especializada en este campo y además que nos responda de forma clara y amigable, la cabecera a enviar por defecto es la siguiente, dependiendo del idioma con el que se quiera interactuar con el chat.

Cabecera en inglés:

The following is a conversation with a computer AI. The AI is helpful, creative clear and very friendly

Cabecera en español:

Lo siguiente es una conversación con una AI. La AI es útil, creativa, clara y muy amigable

Figura 4.1: Selección de cabecera desde la interfaz con /config

Con las frases de contexto, podemos enseñar a GPT-3 a responder realizando tareas de clasificación, completar texto, búsqueda o mantener un diálogo. Para la aplicación de chat, lo que permite enseñar al motor a mantener un diálogo es principalmente adjuntarle algunas líneas previas a modo de ejemplo, con un prefijo cada una, que permita diferenciar lo que es la pregunta de la respuesta. En este chat, el prefijo a enviar junto con la pregunta es "*HUMAN: "*y tras la pregunta, se adjuntará la palabra *"AI: "*. Con esto, GPT-3 sabrá que le toca responder. Para verlo más claro, muestro las frases de contexto configuradas en la aplicación:

Frases en inglés:

HUMAN: Hi! Who are you?
AI: I am an AI who Will help you solve your computer questions
HUMAN: I have many questions
AI: What do you want to ask me?

Frases en español.

HUMAN: Hola, ¿Quién eres?
AI: Soy una AI que te ayudará a resolver tus preguntas sobre informática
HUMAN: ¿Me puedes contestar siempre en español?
AI: Por supuesto, ¿Qué me quieres preguntar?

Como se puede apreciar, en el caso de las frases en español, se pide explícitamente a GPT-3 que nos responda en español. Esto ayuda a que el motor responda realmente en este idioma, y aunque los modelos han sido entrenados mayoritariamente en inglés, es tal la potencia, que son capaces de no solo responder, sino incluso de traducir cualquier texto en una gran cantidad de idiomas diferentes,

aunque esta característica solo lo hace bastante bien en las versiones más potentes de los modelos.

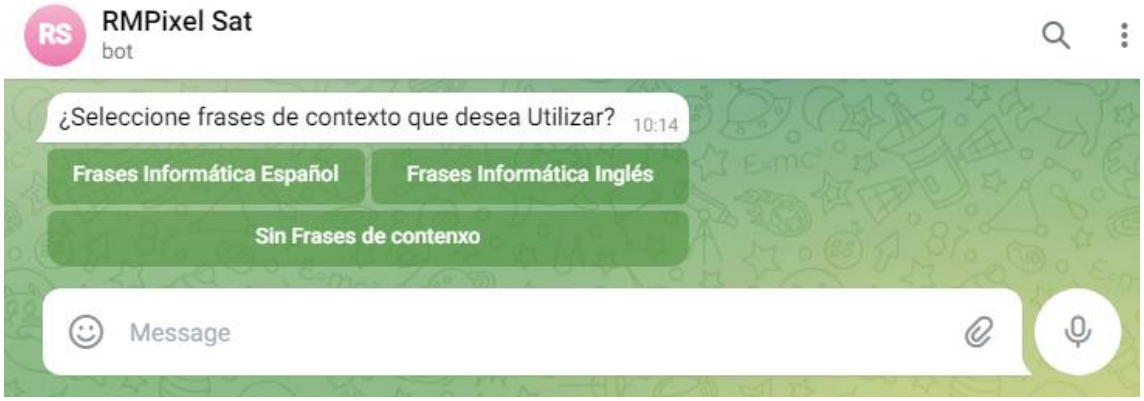

Figura 4.2: Selección de frases de contexto predeterminadas.

4.2 Creación fichero train.jsonl.

El fichero *train.jsonl* es un fichero creado por la aplicación ya en formato *Fine-tuning* (ver figura 2.8) que se utiliza para el reentrenamiento de los modelos GPT-3. Este fichero se genera desde la aplicación en cuanto se le solicita mediante la interfaz de configuración gráfica a través del comando */config.(ver figura 4.1)*

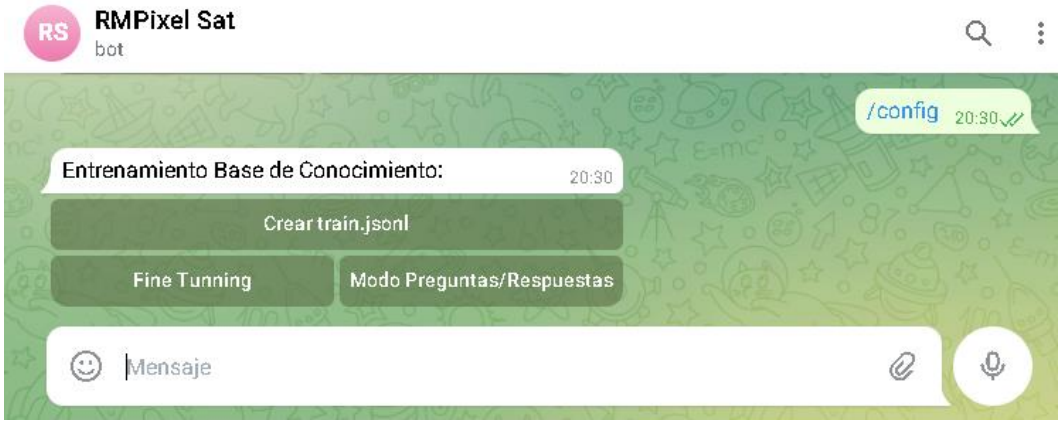

Figura 4.3: Menú para reentrenamiento GPT-3

La creación de este fichero se consigue obteniendo los tweets desde una cuenta específica de Twitter. Como el entrenamiento del modelo se basa en preguntas y respuestas, es importante, obtener los tweets desde cuentas que actúan generalmente respondiendo a preguntas iniciadas por usuarios. Es por esto, y teniendo en cuenta que la especialización usada en la aplicación es de tipo informático, que la cuenta de twitter utilizada por defecto es @MicrosoftAyuda. Esta cuenta generalmente responde a preguntas realizadas por los usuarios, aunque también puede simplemente publicar tweets sin pregunta previa. Para la programación de la API de Twitter se utiliza la librería de la comunidad Tweepy tal y como se indicó en el apartado 3.2.

Dada la naturaleza de este tipo de cuentas, ya que funcionan como un chat, es muy interesante poder seguir el hilo de la conversación. Esto con la versión 1.1 de la API de Twitter, era muy complejo de conseguir, pero en la versión 2.0, se implementa la opción de poder obtener el hilo de una conversación completa, lo que facilita mucho la tarea. Este ha sido el principal motivo por el que se ha elegido esta versión de la API. La secuencia de pasos de la aplicación es la siguiente:

1. Se conecta a Twitter utilizando la *key bearer_token*, esto consigue autenticación de aplicación OAuth 2.0. Se utiliza la clase *Client*, que es la que da acceso a la API 2.0.

```
client = tweepy.Client(bearer_token=xxxxxxxxxx)
```

Figura 4.4: Conexión a API 2.0 Twitter.

2. Se realiza una primera búsqueda de los tweets recientes de la cuenta especificada. En este caso, como se ha mencionado es @MicrosoftAyuda. El método *search_recent_tweets()* hace uso de la API *recent searches* de Twitter, la cual está limitada a la búsqueda de tweets de los últimos 7 días. Si se llega al límite establecido por Twitter, el programa esperará el tiempo necesario para poder seguir realizando peticiones.

   Se solicita en la búsqueda que se obtengan, entre otros, los campos *conversation_id y in_reply_to_user_id* de cada tweet. Además, se eliminan de la búsqueda los *retweets*, ya que no nos interesan.

```
query = 'from:MicrosoftAyuda -is:retweet'

tweets = client.search_recent_tweets(query=query, tweet_fields=['conversation_id',
'created_at',"attachments","author_id","lang","in_reply_to_user_id","referenced_tweets"
], max_results=100)
```

Figura 4.5: Consulta base para la obtención de tweets.

3. Ahora se dispone de una lista de tweets sin clasificar, sin saber cuáles son preguntas y cuáles son las respuestas.

   Para solventar esto, dentro de un bucle comparamos cada tweet con el valor de su variable *in_reply_to_user_id*. Si esta variable no es nula, significa que es un tweet en respuesta a algún usuario, por lo tanto, miembro de un hilo de una conversación de al menos dos tweets. Así que usando el campo *conversation_id* (que representa un código único compartido por todos los tweets de un mismo hilo), realizamos una nueva búsqueda, pero esta vez solo de los tweets de esa conversación en concreto, que serán los que compartan el mismo número en *conversation_id*, devolviéndonos una lista con el hilo de la conversación completa.

```
    if tweet.in_reply_to_user_id != None:
        query = 'conversation_id:' + str(tweet.conversation_id)
        hilo = client.search_recent_tweets(query=query, tweet_fields=['conversation_id',
'created_at',"attachments","author_id","lang","in_reply_to_user_id","referenced_tweets"],
max_results=10,expansions="referenced_tweets.id")
```

Figura 4.6: Obtención de un hilo completo si el tweet es una respuesta.

4. Si en la lista anterior se encuentra algún tweet que contenga imágenes o sonidos, se descarta la conversación completa.

5. Una vez el hilo se da por válido, se prepara el fichero train.jsonl siguiendo el formato para *Fine-tuning.* Se presupone que el primer tweet es una pregunta y los siguientes seguirán la secuencia respuesta, pregunta, respuesta…

6. Esta búsqueda de hilos de conversaciones se realiza para cada tweet con un *conversation_id* distinto.

4.3 Funcionamiento general

Un resumen general de los pasos principales que realiza la aplicación, son los siguientes:

1 Aplicación ejecutándose en *modo polling*
La aplicación comprueba continuamente la llegada de mensajes nuevos al *bot* ***RMPixel_bot.*** En caso afirmativo, envía el mensaje y el identificador único del usuario emisor del mensaje a una función en concreto dependiendo de si es un comando, un comando desconocido o un mensaje cualquiera.

2 (Opcional) En caso de tener activada la traducción automática con ***/translate*.**
Esta característica permite traducir la entrada del usuario al idioma inglés. Se consigue enviando el texto a *Google Translate*. Esto permite obtener mejores respuestas, ya que, como se ha dicho anteriormente, los modelos han sido entrenados mayoritariamente en inglés.

3 Se formatea el *prompt* a enviar utilizando técnicas de *few-shot learning*.

La entrada del usuario, tal vez ya traducida al inglés si estaba activada la opción, se le antepone el prefijo *"HUMAN: "* seguido del texto del usuario y finalmente el sufijo en una nueva línea *"AI: ".* O sea, para la entrada por parte del usuario *"Mi ordenador no arranca, ¿cómo puedo solucionarlo?"* el *prompt* formateado sería:

"HUMAN: Mi ordenador no arranca, ¿cómo puedo solucionarlo?
AI: "

4 Se actualizará la lista de frases de contexto agregando la nueva entrada al final, borrando la más antigua en caso de que con esta nueva frase se exceda el valor del número máximo establecido con el parámetro /memory.

5 Dependiendo de la configuración establecida, se agregará la cabecera al *prompt* antes de ser enviado.

6 Enviar *prompt completo (cabecera + frases de contexto)* al modelo GPT-3 configurado.

7 Recibir la respuesta, llamada *completion.* A esta respuesta la llamaremos **respuesta limpia.**

8 Se crea una ***respuesta formateada*** formada por el prefijo *"AI : "* más la respuesta limpia del punto anterior.

9 (Opcional) En caso de tener activada la traducción automática.
Esta característica permite traducir la *respuesta limpia* obtenida en el punto 7, al idioma español. Se consigue enviando el texto a Google Translate.

10 Devolver la *respuesta limpia,* tal vez traducida, a través de la interfaz Telegram al usuario que realizó la petición.

11 Se actualizará la lista de frases de contexto agregando la *respuesta formateada* creada en el punto 8, borrando la más antigua en caso de que con esta nueva frase se exceda el valor del número máximo establecido con el parámetro /memory.

El esquema de funcionamiento de la aplicación, una vez implementada la misma en un equipo, queda tal y como refleja la figura 4.1.

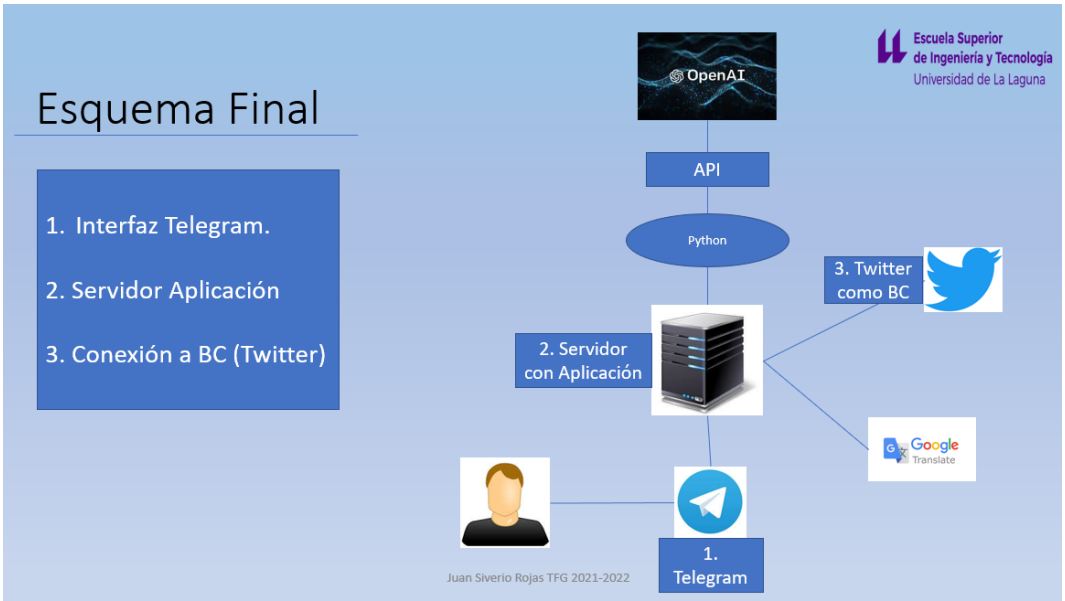

Figura 4.1 Esquema de funcionamiento aplicación servidor.

En el apartado 3.4.1, ya se mostraron algunos comandos con los que modificar parte del comportamiento de GPT-3 desde la aplicación. En la tabla 3, se muestra un listado completo de los comandos disponibles desde la interfaz, con una pequeña descripción cada uno. Este listado se obtiene mediante el comando */help*.

Comando	Descripción
/help	*Muestra esta ayuda*
/start	*Mensaje introductorio al manejo del bot*
/restart	*Reinicia el estado del Bot a un estado limpio. (El contexto, basado en preguntas y respuestas anteriores se pierde)*
/end	*Finaliza la ejecución del Bot. Deberá ser iniciado manualmente en el servidor*
/debug	*Activa o desactiva el modo depuración. Se mostrarán tanto por consola en el servidor como en la interfaz Telegram al propio usuario, varios mensajes de depuración, así como las respuestas en crudo de los motores de GPT-3.*
/status	*Muestra la configuración actual*

Comandos exclusivos para el motor GPT-3	
/temperature	*Especifica lo aleatorio de las respuestas. Rango (0 preciso - 1 más aleatorio)*
/top_p	*Alternativa a la temperatura. Indica el porcentaje de respuestas con mayor puntuación considerados Rango (0 - 1)*
/n	*Número de respuestas devueltas por el motor. Permite a mi algoritmo seleccionar la mejor. ¡Ojo!, consume más tokens.*
/max_tokens	*Número máximo de tokens en las respuestas*
/translate	*Activa o desactiva la auto traducción ES-EN <-> EN-ES*
/memory	*Cantidad de frases de contexto a enviar junto con la pregunta y cabecera si se incluye.*
Comando para configuración gráfica.	
/config	*Opciones de configuración*

Tabla 3: Listado de comandos obtenidos con /help.

4.4 Resultados

Tras realizar numerosas pruebas, se obtiene que en general, GPT-3 muestra un comportamiento destacado en lo que a generación de texto automático se refiere. Estos modelos del lenguaje han sido entrenados con una gran cantidad de datos tomados de internet sin etiquetar, por lo que tiene sentido que, dada una entrada, pueda predecir y generar texto adicional gramatical y ortográficamente casi perfecto.

Sin embargo, en lo que respecta al funcionamiento tipo *chat*, muestra muchas carencias, sobre todo en lo referente a la semántica. Parece que GPT-3 aunque desconozca la respuesta, siempre intenta responder, aunque sea incorrectamente. Básicamente al no estar estos modelos diseñados para el diálogo, tiene que adivinar lo que está pasando. No obstante, al utilizar pistas, o sea, la técnica del *few-shot learning,* se consigue una mejora increíble, obteniendo unos diálogos mucho más creíbles.

La mejora obtenida con el *Fine-tuning,* ha sido muy decepcionante, aunque con un buen fichero de entrenamiento, seguramente mejoraría bastante ya que tras analizar el fichero *train.jsonl* manualmente, se comprueba que hay mucha basura. Por ejemplo, se encuentran respuestas que ocupan más de un tweet, por lo que la aplicación, puede entender esos tweets continuación del anterior como una nueva pregunta o respuesta. Además, el número de tweets con imágenes es bastante alto, ya que los usuarios suelen

aportar capturas o videos explicativos. Una variante que se ha utilizado fue probar cogiendo solo la primera pregunta y una única respuesta del hilo de conversación, intentando así, eliminar mensajes poco precisos propios de una conversación continuada y alargada en el tiempo, sin embargo, esto no mejoró los resultados. El hecho de que el *Fine-tuning* solo pueda aplicarse a los modelos base, con menor cantidad de datos y más antiguos que los nuevos modelos *instruct,* también es una limitación.

Por otra parte, el método de funcionamiento de preguntas / respuestas*(answers)* que se está implementando en los modelos base de GPT-3 y que se encuentra aún en versión beta, parece muy prometedor, ya que es capaz de extraer respuestas de las propias preguntas.

Un aspecto importante que mejora mucho la obtención de respuestas adecuadas es el uso de la traducción automática aplicando el comando */translate,* ya que, ante las mismas preguntas, las respuestas teóricamente más adecuadas se obtienen en mayor proporción cuando se realizan en inglés, o en su defecto, con la traducción automática.

Las pruebas se han realizado con diferentes configuraciones, motores, traducción automática activada o desactivada, diferentes frases de contexto y cabeceras, y muchas variaciones más en los parámetros. Todas estas configuraciones se han probado con las mismas preguntas, la cuales han estado relacionadas con posibles incidencias informáticas que podría reportar un usuario. Se han utilizado preguntas bastante empíricas, para poder comprobar el grado de precisión de las respuestas.

Capítulo 5
Conclusión y Líneas futuras

A vista de los resultados, se observa que GPT-3 como modelo del lenguaje para establecer diálogos, sigue estando bastante limitado, aunque con una buena especialización, seguramente se puede conseguir algo bastante funcional. El problema es que esta especialización sigue dependiendo de un fichero de entrenamiento muy bien preparado, a veces, difícil de conseguir si no se hace manualmente.

También se demuestra que hay grandes diferencias entre los diferentes motores, ya que todos funcionan bastante bien en inglés, pero solo el más potente puede responder de forma más o menos correcta en otros idiomas.

La principal línea de trabajo futuro que se extrae de los resultados vistos anteriormente sería la investigación de un mejor método de *Fine-tuning,* centrándose en el modo de creación y /u obtención de un mejor fichero de entrenamiento, mediante un tratamiento mejorado de los *tweets,* aplicando técnicas más exhaustivas de tokenización *y* depuracion de la información. Adicionalmente, se podría conseguir una suscripción a Twitter de tipo *Academic Research,* para poder recuperar una mayor cantidad de información y realizar un reentrenamiento más profundo. También sería interesante profundizar en un algoritmo que identifique mejor que *tweets* de los hilos de conversación son preguntas y cuales son respuestas

Otra importante línea de investigación debe ser el modo *answers,* que está consiguiendo excelentes resultados, modo con el que sería posible una especialización superior, aunque renunciando a la generalización que se consigue realizando *Fine-tuning* a un modelo ya entrenado.

Como valoración final, parece que se ha dado un salto importante en el NLP con la aparición de los *Transformers*, aun así, seguimos lejos de modelos con una lógica y sentido común más humanos.

5.1 Conclusion

Based on results GPT-3, as a language model for dialogs seems to be still quite limited, although with a good specialization, it should be quite functional The problem is that this specialization still depends on a very well prepared training file, sometimes difficult to achieve if it is not done manually.

Results also show big variations between the different engines. They all work quite well in English, but only the most powerful can respond more or less correctly in other languages.

According to the results described above, the main line of future work should be the investigation of a better Fine-tuning method, focused on the way of creating a better training file, through an improved treatment of the tweets, applying more exhaustive

techniques of tokenization and debugging of the information. Additionally, an Academic Research-type Twitter subscription could be obtained, in order to retrieve a greater amount of information and carry out a more in-depth retraining. It would also be interesting to delve into an algorithm that better identifies which tweets in the conversation threads are questions and which are answers.

Another important line of future research should be the answers mode, which is achieving excellent results. This is a mode with which a very good specialization would be possible, although renouncing the generalization that is achieved by Fine-tuning an already trained model.

As a final assessment, it seems that an important leap has been made in the NLP with the appearance of the Transformers, even so, we are still far from models with more human logic and common sense.

Capítulo 6
ibliografía

[1] Web oficial OpenAI (2022). https://openai.com/api/

[2] Web oficial Telegram (2022). https://core.telegram.org/api

[3] Librería Python-telegram-bot (2022). https://docs.python-telegram-bot.org

[4] Twitter (2022). https://developer.twitter.com/en/docs

[5] Librería Tweepy (2022). https://docs.tweepy.org/en/stable/

[6] Wikipedia. https://www.wikipedia.org

[7] InteractiveChaos. (2022) https://interactivechaos.com/es/manual/tutorial-de-machine- learning

[8] Ivan Pinar Domínguez. Procesamiento del lenguaje Natural con Python(NLP). Udemy 2,5 horas, 2022

[9] Martin Jocqueviel. Procesamiento del Lenguaje Natural Moderno en Python. Udemy 10 horas, 2021

[10] María Santos, Frogames Support Team, Curso completo de Python 3 de la A a la Z, Udemy 55 horas, 2022

[11] Juan Gabriel Gomila, Frogames Support Team,Curso completo de Machine Learning: Data Science en Python, Udemy 48 horas, 2021

[12] Youtube 2022, Dot CSV, https://www.youtube.com/c/DotCSV

[13] Youtube, 2022, Xpikuos, https://www.youtube.com/c/Xpikuos

[14] Youtube, 2022, Codificando Bits, https://www.youtube.com/c/codificandobits

[15] Youtube, 2022, Spain AI, https://www.youtube.com/c/SpainAI

[16] Aurélien Géron, Aprende Machine Learning con Scikit-Learn, Keras y TensorFlow, O'Reilly Meida, ANAYA MULTIMEDIA, 2020

[17] Rajat Tripathi, Pinecone, 2020. Libre.

[18] Li, Dingcheng & Zhang, Jingyuan & Li, Ping. (2019). TMSA: A Mutual Learning Model for Topic Discovery and Word Embedding. 10.1137/1.9781611975673.77.

[19] Google, 2017, The Transformer – model architecture, Figura, recuperado de *"Attention is All You Need", página 3*