UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Chatbot basado en *Large Language Models* y técnicas de *Retrieval-augmented Generation* para asistente de dudas sobre la realización del Trabajo Fin de Grado Documentación Técnica.

Presentado por José María Redondo Guerra
en Universidad de Burgos — 15 de enero de 2024
Tutores: José Ignacio Santos Martín, Carlos López Nozal

Índice general

Índice de figuras

Índice de tablas

Listings

Apéndice A

Plan de Proyecto Software

A.1. Introducción

La planificación del proyecto de software para el desarrollo del chatbot representa una fase crucial dentro del ámbito de este Trabajo de Fin de Grado. En esta sección, se abordará la estructuración y programación detallada de todas las actividades que llevaron a cabo la creación del chatbot, desde su concepción hasta su implementación efectiva.

Se llevará a cabo una evaluación minuciosa de la viabilidad de la solución propuesta, considerando aspectos económicos y legales. Se detallará cómo se establecieron plazos y etapas con el objetivo de garantizar un desarrollo eficiente del proyecto dentro del tiempo previsto. Este plan de proyecto, dividido en secciones clave, no solo trazará temporalmente la ejecución de las fases, sino que también realizará un análisis crítico de la viabilidad económica y legal del proyecto, asegurando así su coherencia y conformidad con los requisitos establecidos.

A.2. Planificación temporal

Para el desarrollo de este proyecto se ha seguido mayoritariamente una metodología *Agile* o ágil. La metodología *Agile* es un enfoque de desarrollo de software que se centra en la flexibilidad, la adaptabilidad y la entrega incremental. A diferencia de los enfoques tradicionales de desarrollo de software, que suelen seguir un modelo de *waterfall* o cascada, donde cada fase del proyecto se completa antes de pasar a la siguiente, la metodología

ágil aboga por la colaboración continua entre equipos multifuncionales y la entrega iterativa de software funcional.

Algunos de los marcos de trabajo ágiles más conocidos incluyen *Scrum*, *Kanban* y *Extreme Programming* (XP). Estos marcos proporcionan prácticas específicas y roles que ayudan a implementar los principios ágiles en proyectos concretos. En este caso no se dispone de un equipo de desarrolladores por lo que se ha optado por usar parcialmente el *framework Kanban* para el trabajo entre alumno y tutores.

Gestión del Proyecto

Kanban es una metodología ágil que se originó en el sistema de producción de Toyota y se ha aplicado exitosamente en el desarrollo de software y la gestión de proyectos. Se basa en el principio de visualizar el trabajo, limitar el trabajo en curso y maximizar el flujo de trabajo [1].

La metodología Kanban se ha vuelto popular en entornos donde la demanda de cambio es constante y la capacidad de respuesta rápida es crucial, ya que permite una gestión más ágil y adaptable del trabajo. Y este es precisamente el motivo por el que está muy extendida en proyectos de Software como este TFG.

Para visualizar las tareas se han creado *Issues* en GitHub/Zube.io. Estas *Issues* contienen una descripción del objetivo, las tareas a realizar para conseguir ese objetivo y como validar que se ha conseguido ese objetivo(*Definition of Done*). Se puede ver un ejemplo de una *Issue* en la figura A.1.

Figura A.1: Descripción de una *Issue* en GitHub que contiene Objetivo, tareas y criterios de éxito.

Para ayudar en la gestión del proyecto con la metodología *Kanban*, se ha usado la herramienta Zube.io, explicada en el apartado 4 de la memoria. Se disponen de distintas fases en Zube para mostrar los distintos estados en los que se puede encontrar una *Issue*. Ver figura A.2.

El proceso seguido después de la creación de un *Issue* ha sido el siguiente:

1. **Refinamiento y estimación:** Se define el objetivo, se estima una puntuación de tiempo basado en la complejidad del trabajo a realizar y por último se pasa al *Backlog*.

2. **Planificación en el *Sprint*:** En la reunión quincenal con los tutores se planifica el siguiente *Sprint* y con ello se mueven las *Issues* al *Backlog* del *Sprint* en curso.

3. ***Issues* en proceso:** Siguiendo la metodología *Kanban*, no se han tenido mas de 2-3 *Issues* en proceso al mismo tiempo. Una vez que están acabadas se han pasado a la siguiente fase para ser validadas.

4. **Validación:** Cuando las *Issues* están completas, se ha comprobado que el resultado era el deseado. De no ser satisfactorio se ha devuelto a una fase anterior o se han reformulado los objetivos.

5. ***Review* del *Sprint*:** En la reunión quincenal con los tutores se ha repasado el trabajo realizado en el Sprint anterior y se han archivado las *Issues* ya finalizadas y revisadas.

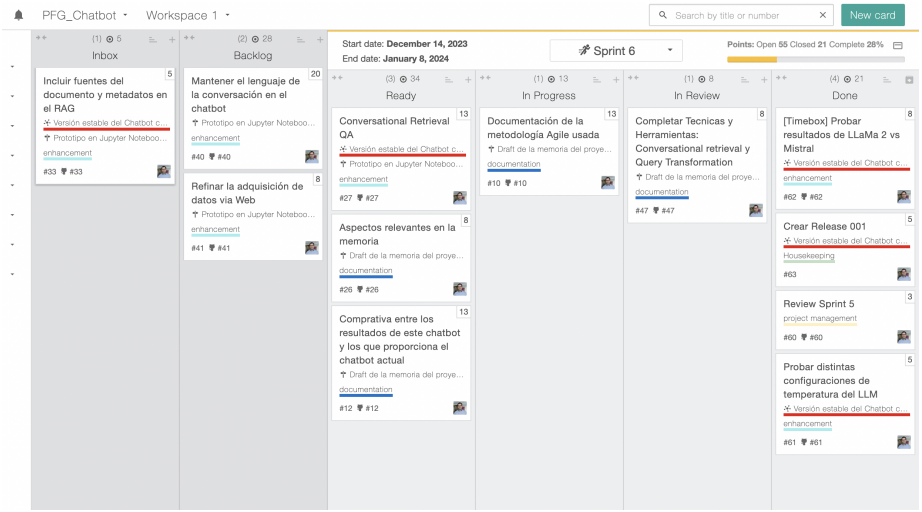

Figura A.2: *Sprint Board* de Zube.io con los distintos estados de las *Issues*.

Milestones

Las *milestones* o hitos en los proyectos de software son eventos o logros significativos que marcan el progreso y el éxito en el desarrollo del proyecto. Estos hitos son puntos clave en el cronograma del proyecto y sirven como indicadores importantes para evaluar el avance hacia los objetivos establecidos. Cada milestone contribuye al logro del objetivo final del TFG y ayuda a mantener el enfoque y la dirección.

- **Kick-off y configuración inicial del proyecto:** Puesta en marcha del TFG. Partiendo de las reuniones iniciales con los tutores se deberá crear la estructura necesaria para el desarrollo del proyecto.
- **Prototipo en Jupyter Notebook de LLM usando datos propios con LangChain:** Creacion de un Notebook de Jupyter que usando LangChain cree un Chatbot usando datos propios a traves de un RAG sobre un LLM.
- **Creación de una interface para el chatbot:** Creación de una UI para que se puede interactuar con el Chatbot desde un entorno mas agradable para el usuario.
- **Borrador de la memoria del proyecto para revisión de los tutores**: Creación de la primera versión de la memoria y los anexos para que los tutores puedan revisar y dar *Feedback*.

- **Material para la entrega final del TFG:** Preparación del material para la entrega del proyecto fin de Grado(memoria final, anexos, presentaciones, vídeos,...).

Organización en *Sprints*

Se han realizado Sprint quincenales que se han planificado y revisado en la reunión quincenal con los tutores del TFG. Las reuniones han servido para, siguiendo la metodología *Agile*, revisar el Sprint anterior, planificar el siguiente y hacer una pequeña retrospectiva para mejorar el trabajo conjunto.

- **Sprint 1(6/10/2023 - 17/10/2023):** El inicio de este sprint lo marcó la primera reunión con los tutores, en la que se dieron las indicaciones de lo que se buscaba con el proyecto y se establecieron los primeros objetivos. Se estableció LaTeX como herramienta para la documentación y GitHub como repositorio. Documentación del *abstract*, Introdución y objetivos.

- **Sprint 2(18/10/2023 - 30/10/2023):** El principal objetivo de este Sprint es tener un prototipo para validar el uso de LLM en Jupyter. En la parte relativa a documentación se pone el foco en el apartado de conceptos generales.

- **Sprint 3(1/11/2023 - 15/11/2023):** Con un prototipo rudimentario pero suficiente para validar los riesgos tecnológicos del proyecto, se pasa a mejorar el cuaderno de Jupyter de Kaggle para que también incluya las técnicas RAG. Se finalizarán lo conceptos generales completando los apartados que faltan.

- **Sprint 4(16/11/2023 - 29/11/2023):** Migración de la versión de Kaggle basada en cuadernos de Jupyter a ficheros en local. Se comienza a desarrollar una metodología para validar los resultados del chatbot a la par que se investigan posibles vías para crear una UI. En la parte de documentación se comienza con el apartado relativo a técnicas y herramientas usadas en el TFG.

- **Sprint 5(30/11/2023 - 14/12/2023):** Se extienden los *Issues* relativos a validación y pruebas. Este apartado lleva más tiempo del esperado y es necesario a mayores estabilizar la versión del Chatbot existente. Por último se crea una UI basada en Streamlit.

- **Sprint 6(15/12/2023 - 7/1/2024):** Con la versión del Chatbot estable, el foco de este Sprint es avanzar en la documentación de la memoria. Se deben completar los apartados Aspectos relevantes y los anexos de la memoria. En el aspecto técnico son necesarias algunas mejoras en los datos de entrada y en la temperatura del LLM.
- **Sprint 7(8/1/2024 - 16/1/2024):** En este último *Sprint* el foco es preparar el material de la entrega. Se efectúan las correcciones en la memoria y Anexos basados en los comentarios de los tutores.

Métricas Ágiles

La herramienta Zube permite de forma sencilla generar gráficas para el seguimiento de las métricas *Agile* mas comunes [12].

Las gráficas *Burnup* son herramientas visuales utilizadas en la gestión de proyectos para mostrar el progreso del trabajo realizado en comparación con las metas y el alcance planificado, ver figura A.3.

Figura A.3: Gráfica *Burnup* del Sprint 3.

A diferencia las gráficas *Burndown* muestran el trabajo restante, ver figura A.4.

Figura A.4: Gráfica *Burndown* del Sprint 3.

También se puedo visualizar la velocidad de los distintos *Sprints*, ver figura A.5. Entendiendo por velocidad el número total de puntos de *issues* que se han completado en el *Sprint*.

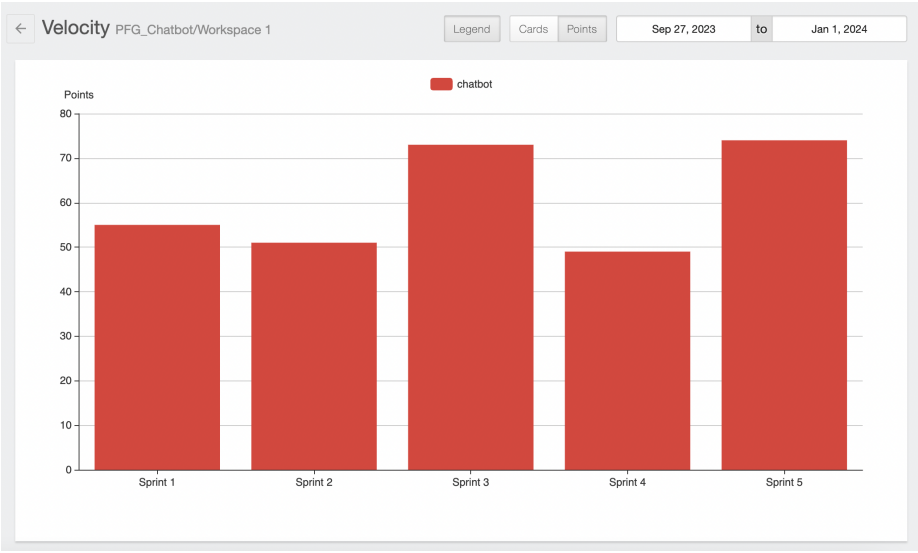

Figura A.5: Velocidad de los primeros 5 *Sprints* basada en los puntos de los *Issues*.

Software	Licencia	Fuente
Python	GPL-compatible	[11]
Misual Studio Code	MIT License	[7]
Streamlit Linbrary	Apache 2.0 license	[15]
LangChain	MIT License	[5]
Mistral B7	Apache 2.0 license	[8]
HuggingFace API	Apache 2.0 license	[4]
Pandas Library	BSD 3-Clause License	[9]
GitHub	MIT License	[3]
Zube	MIT License	[17]

Tabla A.1: Licencias software.

A.3. Estudio de viabilidad

En esta sección, se llevará a cabo un análisis exhaustivo de los costes asociados y los potenciales beneficios del proyecto desde una perspectiva empresarial. Se prestará especial atención a la gestión de recursos, considerando no solo los costes directos, sino también cualquier otro gasto derivado que pueda surgir en el contexto de una operación empresarial real.

Este enfoque integral tiene como objetivo proporcionar una evaluación completa de la inversión requerida para la implementación exitosa del proyecto, así como identificar las oportunidades de retorno y beneficios que puede ofrecer a la organización.

Viabilidad económica

Este apartado va a analizar los costes y posibles beneficios del proyecto vistos desde una perspectiva empresarial; se tendrán en cuenta los recursos humanos, así como todos los gastos derivados que se generarían en una empresa real.

Licencias

En la Tabla A.1 se recoge la información relativa a las licencias software de los programas utilizados en el proyecto.

En el caso de Zube y GitHub la licencia usada es la versión básica que es gratuita y para pequeñas organizaciones es suficiente.

La licencia Apache es libre, simple, sin *copyleft* y permisiva. GNU es libre, abierta y con *copyleft*. MIT License es abierta, son *copyleft* y permisiva.

Como resumen las licencias usadas en estos momento no suponen ni costes ni problemas legales para la explotación del chatbot.

Gastos

En la fase actual de desarrollo de los LLM es difícil de calcular los gastos reales que este chatbot tendría en la fase de explotación. Como se ha indicado anteriormente, la tecnología está lejos de poder ser usada como si fuera un producto estándar y es previsible que se necesite mejorar el desarrollo y actualizar partes del chatbot en la fase de mantenimiento.

El salario mensual bruto de un Ingeniero Informático junior se sitúa en torno a los 24.000€ brutos anuales, por lo que se va a suponer un sueldo mensual de 2.000€ brutos.

Los costes de la Seguridad Social para la empresa son los siguientes: 23,6 % costes comunes, 0,2 % fogasa, 0.6 % formación profesional y una cotización de entre el 5,5-6,7 %, que vamos a suponer del 5,6 % para redondear, sumando en total un 30 %. Para el trabajador la Seguridad Social supone un 4,7 % y el IRPF es de un 12 % para este tramo.

La empresa paga un 30 % de estos 2.000€ a la Seguridad Social. De estos 2.000€ el trabajador paga a la seguridad social un 4,7 % y el IRPF lo que supone 334€ (94€ + 240€). Por lo que el sueldo neto será de 1.666€.

En la Tabla A.2 se calcula el coste humano total.

Concepto	**Coste**
Salario mensual bruto	1.500€
Cotización seguridad social	600€
Total mensual	2.100€

Tabla A.2: Costes humanos.

En la Tabla A.3 se muestra el gasto total en componentes físicos. Se asume un periodo de amortización para el hardware de 3 años (36 meses) y que han sido utilizados 12 meses.

Concepto	**Coste**	**Coste amortizado**
Ordenador portátil	1.000€	333,3€
Smartphone	300€	100€
Ratón	40€	13,3€
Monitor	150€	50€
Teclado	15€	5€
Total	1.505€	501,6€

Tabla A.3: Costes hardware.

La Tabla A.4 recoge el coste total de la conexión a Internet.

Concepto	**Coste**
Internet	30€ /mes
Total	360€

Tabla A.4: Costes de redes y comunicación.

Costes infraestructura

La Tabla A.5 recoge el coste de alquiler aproximado de una oficina pequeña en Burgos. El precio de alquiler del metro cuadrado en Burgos es en el momento de redactar este documento de 10€.

Concepto	**Coste**
Alquiler oficina	500€ /mes
Total	6000€

Tabla A.5: Costes infraestructura.

En la Tabla A.6 se muestra el coste total del proyecto.

Tipo coste	**Coste**
Humano	25.600€
Hardware	501,6€
Redes y comunicación	360€
Infraestructura	6000€
Total	32.461,6€

Tabla A.6: Costes totales.

Beneficios

El cálculo de los beneficios de este proyecto desde un punto de vista de producto es complejo. No es fácil calcular el valor que se podría dar a este chatbot en sí mismo y es más conveniente plantearlo desde el punto de vista de la realización de un proyecto bajo demanda.

Es decir, se plantea el beneficio, no desde el punto de vista de la UBU sino de un empresa de desarrollo de software que sería la encargada de realizar el proyecto y mantenerlo durante los primeros 12 meses.

En la Tabla A.7 se muestra el coste total del proyecto.

Concepto	**Precio**
Desarrollo	35.000€
Mantenimiento	500€ /mes
Gastos asociados	2.500€
Total	49.500€

Tabla A.7: Facturación.

Todos los importes se han considerado sin IVA y en total se obtiene un beneficio antes de impuestos de unos 17.000€.

Viabilidad legal

Los productos basado en IA y en concreto en LLM son muy recientes y el marco regulatorio está siendo definido en estos momentos. En diciembre de 2023 la Unión Europea aprobó la primera ley para regular el uso de IA [10].

La regulación de la Inteligencia Artificial en la Unión Europea mediante el *AI Act*, es la primera legislación integral sobre IA a nivel mundial. Esta ley tiene como objetivo establecer condiciones óptimas para el desarrollo y uso de esta innovadora tecnología, considerando aspectos que van desde la seguridad hasta la sostenibilidad ambiental.

La Comisión Europea propuso en abril de 2021 el primer marco regulatorio de la UE para la IA. Este marco clasifica los sistemas según el riesgo que representan para los usuarios y establece niveles de regulación correspondientes.

El Parlamento Europeo busca garantizar que los sistemas de IA utilizados en la UE sean seguros, transparentes, rastreables, no discriminatorios y respetuosos con el medio ambiente. Se aboga por la supervisión humana de los sistemas de IA para prevenir resultados perjudiciales.

El Acta de IA introduce diferentes reglas según los niveles de riesgo:

1. **Riesgo Inaceptable:** Se prohíben los sistemas de IA que representen una amenaza para las personas, como la manipulación cognitivo-conductual y la clasificación social basada en comportamientos o características personales.

2. **Riesgo Elevado:** Los sistemas de IA que afecten negativamente la seguridad o los derechos fundamentales se consideran de alto riesgo y se dividen en dos categorías, incluyendo aquellos utilizados en productos regulados por la legislación de seguridad de la UE y aquellos en áreas específicas que deben registrarse en una base de datos de la UE.

3. **Riesgo Limitado:** Los sistemas de IA de bajo riesgo deben cumplir con requisitos mínimos de transparencia que permitan a los usuarios tomar decisiones informadas.

La inteligencia artificial generativa, como ChatGPT, estará obligada a cumplir con los requisitos de transparencia, que incluyen:

- Revelar que el contenido fue generado por Inteligencia Artificial.
- Diseñar el modelo para evitar la generación de contenido ilegal.
- Publicar resúmenes de datos con derechos de autor utilizados para el entrenamiento.

Mientras que este chatbot cumpla con esos aspectos, no debería ser un problema usar esta tecnología en producción desde un punto de vista legal. Parte de estos aspectos deben ser implementados por nuestro chatbot y otra parte deben ser provistos por Mistral como empresa propietaria del LLM. Al ser Mistral una empresa europea con sede en Paris, este aspecto será mas viable que si se hubiese elegido un modelo de una empresa americana.

Apéndice B

Especificación de Requisitos

B.1. Introducción

En esta sección, se exponen los objetivos generales del proyecto junto con los requisitos y su correspondiente especificación. Se realiza un análisis exhaustivo tanto de los requisitos funcionales como de los no funcionales.

- **Requisitos funcionales:** Estos se refieren a los comportamientos específicos que el sistema debe exhibir y están directamente vinculados con los casos de uso.
- **Requisitos no funcionales:** En esta categoría se detallan los criterios, restricciones y condiciones que el cliente impone al proyecto, estableciendo pautas que no están directamente relacionadas con comportamientos específicos, sino con características más amplias del sistema.

B.2. Objetivos generales

- Diseño y desarrollo del chatbot basado en LLM: El primer objetivo es diseñar y desarrollar un chatbot que utilice *Large Language Models* para responder a las preguntas y dudas de los estudiantes en relación con la realización de sus Trabajo de Fin de Grado en Ingeniería Informática.
- Integración de técnicas de RAG para mejora de la precisión: El segundo objetivo es integrar técnicas de *Retrieval-augmented Generation* en

el chatbot. Estas técnicas se utilizarán para mejorar la precisión y relevancia de las respuestas del chatbot, aprovechando la información contenida en documentos FAQ, histórico y reglamento del TFG.

- Comparación de rendimiento con chatbot preexistente: El último objetivo implica comparar el rendimiento y la eficacia del chatbot desarrollado en este proyecto con oun chatbot preexistente que fue creado en un proyecto de fin de grado anterior. Esto permitirá identificar las mejoras y ventajas de la nueva aproximación.

B.3. Catálogo de requisitos

Requisitos funcionales

- **RF-1 Interacción texual:** la aplicación debe permitir al usuario interactuar con ella mediante la introducción de texto.
- **RF-2 Procesamiento del lenguaje natural:** la aplicación debe ser capaz de reconocer las preguntas que introduce el usuario y procesarlas para encontrar información relevante.
- **RF-3 Formulación de respuestas:** el chatbot ha de ser capaz de formular respuestas en lenguaje natural con información relevante a la pregunta.
- **RF-4 Información mediante hipervínculos:** la aplicación debe adjuntar como hipervínculos las redirecciones a otras páginas.
- **RF-5 Informe de error:** la aplicación debe informar de un error al usuario cuando su mensaje de entrada no sea un texto válido o no haya sido posible interpretarlo.
- **RF-6 Iniciar y cerrar la interfaz conversacional:** la aplicación debe permitir iniciar la interfaz conversacional.
- **RF-7 Entrenar al chatbot:** se puede actualizar los datos del bot para que se mejoren las respuestas con nuevos datos.

Requisitos no funcionales

- **RNF-1 Rendimiento:** la aplicación tiene que tener un tiempo de respuesta bajo.

- **RNF-2 Usabilidad:** la aplicación debe ser intuitiva y fácil de entender y utilizar.
- **RNF-3 Imagen corporativa:** la aplicación debe mantener los colores y estética corporativos de la UBU.
- **RNF-4 Disponibilidad:** la aplicación debe estar disponible el mayor tiempo posible.
- **RNF-5 Mantenibilidad:** la aplicación debe ser fácilmente modificable.
- **RNF-6 Portabilidad:** la aplicación debe poder ejecutarse en distintas plataformas.
- **RNF-7 Compatibilidad de navegadores:** la aplicación debe ser compatible para los navegadores más importantes.

B.4. Especificación de requisitos

Diagrama de casos de uso

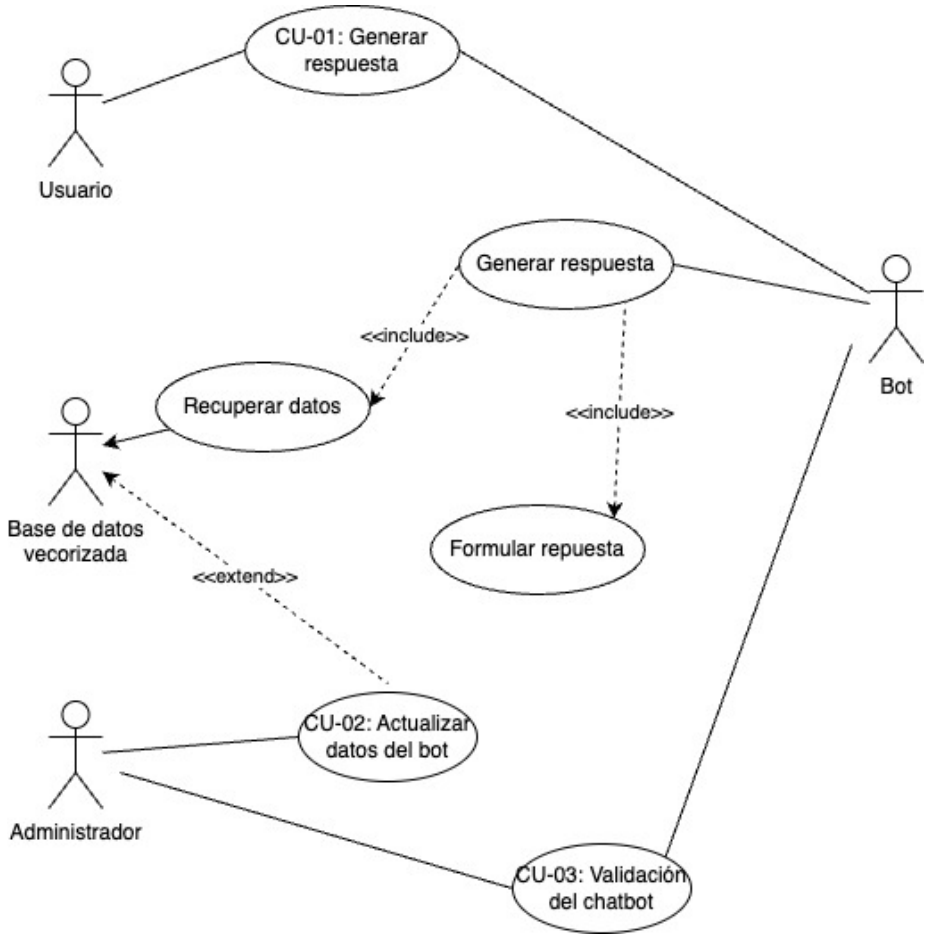

Figura B.1: Diagrama de casos de uso del chatbot.

El actor "usuario" va a ser el estudiante, profesor o cualquier persona con acceso a la asignatura que comience una nueva sesión con el chatbot. El actor "administrador" es el encargado de actualizar los datos para el RAG, ver figura B.1.

Casos de uso

CU-01	**Generación de respuestas.**
Requisitos asociados	RF-1, RF-2, RF-3, RF-4, RF-5
Descripción	El usuario introduce de manera textual una pregunta al chatbot.
Precondición	El chatbot está iniciado.
Acciones	1. El programa analiza la pregunta introducida y recupera información relevante. 2. Se genera una respuesta en lenguaje natural usando la pregunta y la información relevante recuperada. 3. Se responde al usuario en modo texto y por medio de la interfaz conversacional a su pregunta. 4. El chatbot queda a la espera de recibir nuevas preguntas.
Postcondición	Se devuelve un mensaje con la respuesta a la pregunta.
Excepciones	Si el chatbot no es capaz de reconocer la pregunta introducida por el usuario o está mal formulada se informa al usuario por medio de un mensaje de que no ha sido posible entenderle.
Importancia	Alta

Tabla B.1: CU-01 Generación de respuestas.

CU-02	**Actualizar datos del bot.**
Requisitos asociados	RF-2, RF-7
Descripción	El administrador actualiza los datos del chatbot para que la información sea mas relevante.
Precondición	Se han actualizado los datos de entrenamiento en la carpeta correspondiente.
Acciones	1. Por línea de comandos se ejecuta el programa de actualización de datos de entrenamiento. 2. El programa genera una nueva base de datos con la información vectorizada de los documentos FAQ, histórico y reglamento del TFG. 3. Se guarda la nueva base de datos reemplazando a la anterior existente.
Postcondición	Se devuelve un mensaje de éxito.
Excepciones	Si el programa no es capaz de actualizar la base de datos, se muestra un mensaje de error.
Importancia	Alta

Tabla B.2: CU-02 Actualizar datos del bot.

CU-03	**Validación del chatbot.**
Requisitos asociados	RF-1, RF-2, RF-3, RF-4, RF-5, RF-6
Descripción	El administrador verifica que el funcionamiento del chatbot es el adecuado y que las respuestas son parecidas a la información con la que ha sido entrenado.
Precondición	Se ha actualizado la base de datos y la lista de preguntas a verificar.
Acciones	1. Por línea de comandos se ejecuta el programa de validación del chatbot.. 2. El programa ejecuta una serie de preguntas al chatbot sin iniciar la interfaz gráfica y compara la respuesta obtenida con la esperada. 3. Se guarda en un archivo la configuración actual, la pregunta realizada y las respuestas obtenidas y esperadas.
Postcondición	Se ha cerrado correctamente el programa y se ha generado el archivo con la información de la prueba.
Excepciones	Si el programa no es capaz de finalizar el programa de validación, se muestra un mensaje de error.
Importancia	Alta

Tabla B.3: CU-03 Validación del chatbot.

Apéndice C

Especificación de diseño

C.1. Introducción

En esta sección, se abordan detalladamente las especificaciones de diseño del proyecto, centrándose en la creación de un chatbot basado en *Large Language Models* y *Retrieval-augmented Generation*. Las especificaciones de diseño constituyen un componente crucial para el desarrollo eficiente y efectivo del chatbot, definiendo de manera precisa cómo se estructurarán y funcionarán sus distintos elementos. Se exploran aspectos tanto técnicos como funcionales, proporcionando un marco sólido que orientará la implementación del chatbot a lo largo del proyecto.

C.2. Diseño de datos

Se disponen de tres documentos con información relevante acerca de los TFG del grado de ingeniería informática en la UBU. Se dispone de la normativa de Trabajo de Fin de Grado en PDF, un documento con preguntas y respuestas tipo FAQ en formato DOCX y el histórico de proyectos en formato CSV.

Se han realizado unas pruebas integrando en el chatbot con RAG los datos sin preprocesar y los resultados no han sido buenos. La información se encuentra en distintos formatos *.docx*, *.csv* y *.pdf* y contiene comentarios, tablas, pies de página y texto en párrafos. Todo ello hace el proceso de realizar *embeddings* no sea efectivo y no se pueda recuperar despues información relevante durante el RAG.

FAQ Online

Este documento estaba creado para el Chatbot que usaba DialogFlow para la creación de la aplicación [16]. El documento original era un DOCX que contenía tanto texto, como tablas y comentarios. Se puede encontrar en el siguiente enlace: https://github.com/jrg1013/chatbot/blob/main/datasets/ListadoPreguntas-Respuestas%20-%20ONLINE.docx.

La información en el FAQ está segmentada y en el proceso de creación de los *Embeddings*, al no tener una estructura definida los *chunks* no mantenían la estructura semántica. Es cierto que el Chatbot respondía algunas preguntas correctamente al exportar el documento original a TXT de forma automática, pero parte de las preguntas y respuestas se mezclaban al no separarse correctamente.

Se han exportado los datos a formato CSV y se han estructurado mejor. Se puede encontrar el nuevo archivo en el siguiente enlace: https://github.com/jrg1013/chatbot/blob/main/project-app/documents/Preguntas-Respuestas%20-%20ONLINE.csv.

Al usar el *Data Loader* de LangChain para CSV se ha especificado como fragmentar la información correctamente y cómo interpretar cada columna. Esto ha supuesto una mejora considerable en los resultados del Chatbot y en general la información se recupera adecuadamente de la base de datos vectorial.

Histórico de TFGs

Este documento no es usado en el chatbot anterior y se encuentra en formato CSV. Contiene una lista de los TFG realizados en la modalidad *online* en los últimos años e información sobre cada uno de esos proyectos como es el nombre de los tutores o el enlace al repositorio. Se puede encontrar en el siguiente enlace: https://github.com/jrg1013/chatbot/blob/main/datasets/TFGHistorico.csv.

En el histórico de TFG, se ha mantenido el formato CSV, pero tras varias pruebas, ha sido necesario organizar la información en las celdas de forma adecuada y eliminar algunas columnas para evitar problemas. Se puede encontrar el nuevo archivo en el siguiente enlace: https://github.com/jrg1013/chatbot/blob/main/project-app/documents/TFGHistorico.csv.

Principalmente los problemas han venido con el *Data Loader* de LangChain para CSVs. Al leer los datos estos deben estar bien estructurados o los *chunks* de información carecerán de sentido y no se recuperará información

del RAG. Tras varios intentos se ha logrado realizar recuperación de información reduciendo el tamaño de los *chunks* de los *embeddings* y reduciendo el valor de similitud mínimo.

Reglamento para TFG y TFM de la UBU

El último documento usado para el entrenamiento del chatbot ha sido el PDF que contiene el reglamento para TFG de la UBU. Se puede encontrar en el siguiente enlace: `https://github.com/jrg1013/chatbot/blob/main/datasets/reglamentp_tfg-tfm_aprob._08-06-2022.pdf`.

El documento del reglamento de la UBU presenta dificultades a la hora de importar la información en la base de datos vectorial. Principalmente tiene el problema de la estructura de la información con pies de página y encabezados en cada hoja que dificultan la creación de trozos de información coherentes.

El segundo problema es que estamos combinando distintos tipos de *embeddings*. Uno más estructurado proveniente de CSV y otro con lenguaje natural desde un PDF. No es una situación ideal que se intentará manejar en el proceso de recuperación de la información y la parte generativa a través del *prompt*. Este documento no ha sido tratado y se usa el *Data Loader* PyPDFLoader.

C.3. Diseño procedimental

CU-01: Generación de respuestas

Desde el punto de visto del usuario el proceso que sigue el chatbot comienza con la apertura del página del chat en un navegador. En este momento se la aplicación renderiza la UI y se queda a la espera que que el usuario realice la primera pregunta.

Hasta este momento no se ha dado ningún paso para generar un LLM o usar una técnica RAG. Este proceso comienza una vez que el usuario ha introducido su primera pregunta. Una vez esa pregunta se hace llegar hasta la lógica de negocio, esta intenta generar una conexión con la API de HuggingFace. Dependiendo de si la conexión tiene éxito o no, la aplicación devuelve un error o empieza a generar la configuración del LLM. Este paso solo tiene lugar al emitir la primera pregunta de una sesión. A partir de esta primera pregunta, la conexión con la API permanece abierta hasta finalizar la sesión.

Una vez establecido el LLM se pasa al RAG. Se hace una búsqueda por similitud en la base de datos vectorial para recuperar la información relevante a la pregunta realizada. Una vez recuperada esta información se genera un *prompt* sumando esta información a la pregunta original [6].

Con el *prompt* y la configuración del LLM, se puede enviar la *query* a HuggingFace. Esta emitirá un error o una respuesta que será transmitida hasta la UI del chatbot.

Todo este proceso se puede ver en el diagrama de secuencia en la figura C.1.

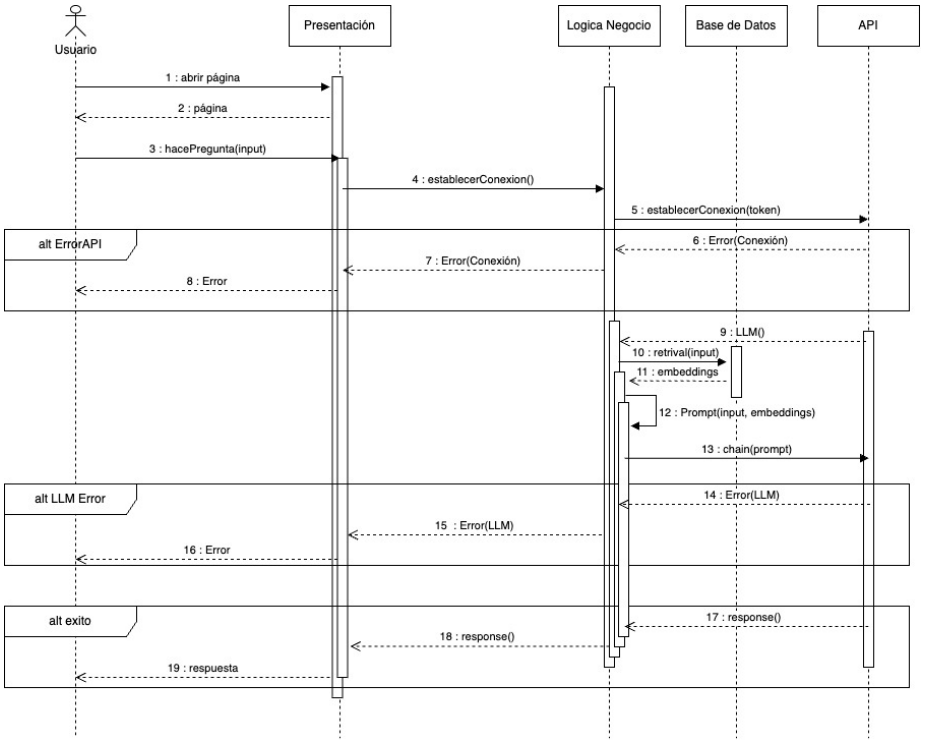

Figura C.1: Diagrama de secuencia CU-01: Generación de respuestas.

CU-02: Actualizar datos del bot

El administrador es el que realiza la actualización de la base de datos vectorial que usa el chatbot para dar respuestas relativas al TFG en la UBU. El proceso comienza con la ejecución del *script learn.py*.

Lo primero que se hace es cargar los archivos de datos con los *data loaders* de LangChain. Una vez que estos datos están en la lógica de negocio, se realiza el corte de la información en *chunks* que se pasan al LLM para que este cree los *embeddings*.

Una vez creados los *embeddings* de la base de datos vectorial se guarda esta base de datos para poder ser usada por el chatbot. Todo este proceso se puede ver en el diagrama de secuencia en la figura C.2.

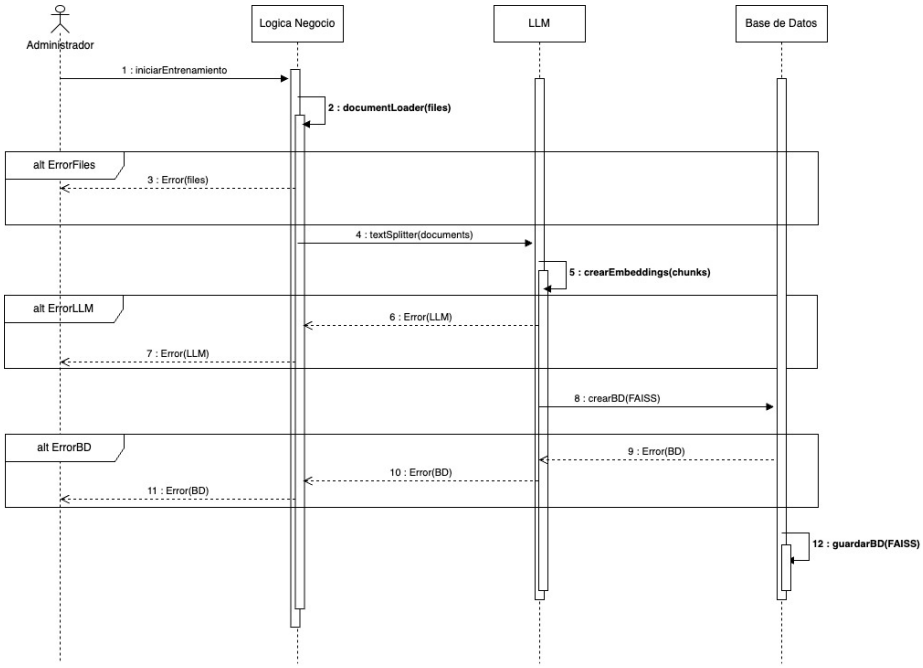

Figura C.2: Diagrama de secuencia CU-02: Actualizar datos del bot.

CU-03: Validación del chatbot

Para realizar la validación del chatbot el administrador ejecuta el *script* preparado para comenzar la secuencia de validación.

Se comienza cargando desde un fichero las preguntas y las respuestas esperadas de dichas preguntas. Se realizan estas preguntas al chatbot y se compara las respuestas dadas con las respuestas esperadas. Para poder comprobar si semánticamente son respuestas equivalentes, se realiza una llamada a un LLM que devolverá una valoración(correcta o incorrecta).

Se guarda la pregunta, las respuestas (tanto la esperada como la generada por el chatbot) y la valoración del LLM se guardan en un fichero de texto para poder comparar distintas configuraciones. Este proceso se puede ver en el diagrama de secuencia en la figura C.3.

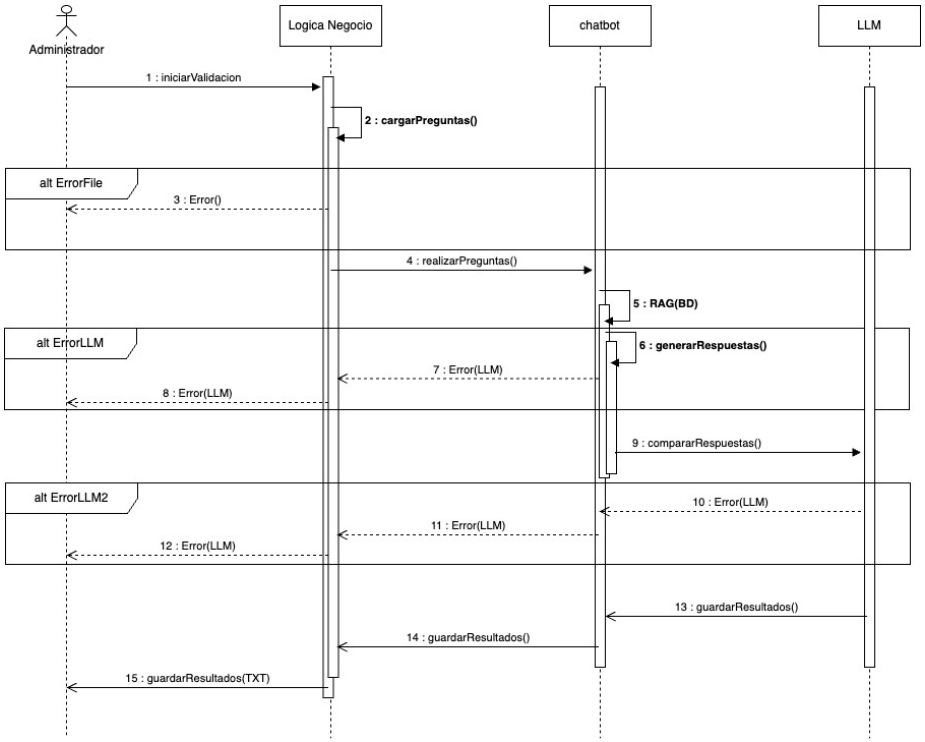

Figura C.3: Diagrama de secuencia CU-03: Validación del chatbot.

C.4. Diseño arquitectónico

Comparado con versiones anteriores del chatbot, este tiene una arquitectura más compleja, ya que no se utiliza un producto disponible, sino que se utilizan distintos recursos y servicios.

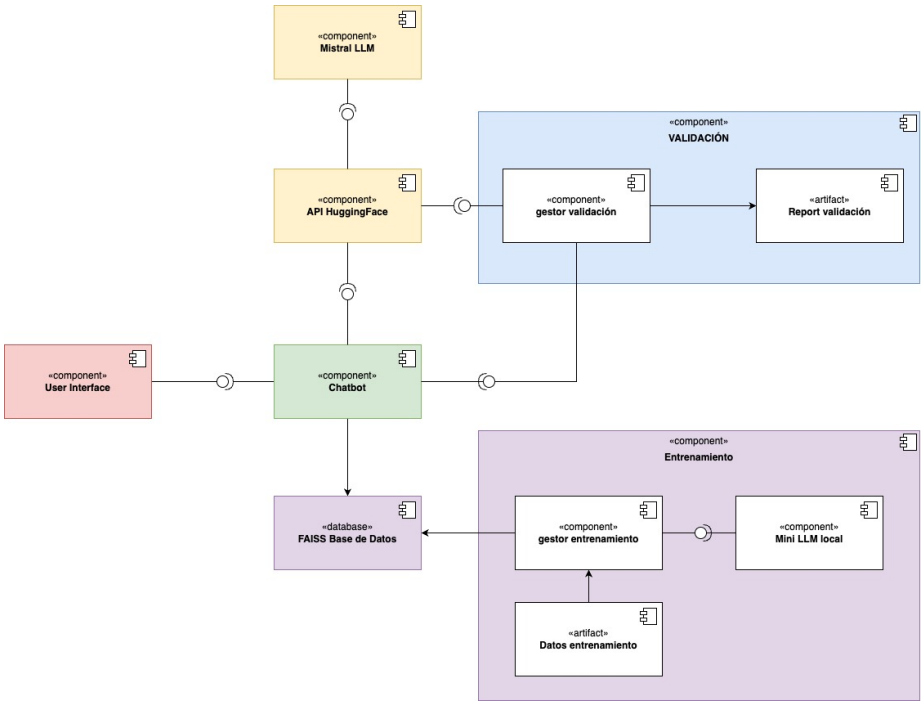

Figura C.4: Diagrama UML de componentes del chatbot.

Como se puede ver en la figura C.5 y en la n la figura C.4, existen principalmente cinco bloques.

- **Base de datos vectorial:** Para la creación de la base de datos se usará LangChain y FAISS, ambos se han explicado en detalle en las secciones correspondientes de la memoria. La principal ventaja de una base de datos vectorial es que permite realizar búsquedas y recuperación de datos rápida y precisa basada en la distancia o similitud de sus vectores. Esto significa que, en lugar de utilizar métodos tradicionales para consultar bases de datos basadas en coincidencias exactas o criterios predefinidos, se puede utilizar una base de datos vectorial para encontrar los datos más similares o relevantes según su significado semántico o contextual.

  Para realizar la base de datos vectorial basada en los datos de las FAQ del Trabajo de Fin de Grado, se crean *embeddings* usando LangChain y separando la información de los CSV en función del atributo que representan. Posteriormente se guarda la base de datos vectorial en memoria y ya estaría lista para ser utilizada por el chatbot.

Este sección se ha separado en un módulo independiente de Python para no tener que crear una base de datos cada vez que se ejecuta el chatbot si no se han añadido datos nuevos. De esta manera el proceso de "entrenamiento" del chatbot y el de ejecución son independientes el uno del otro, lo que mejora el rendimiento y la mantenibilidad del código.

- **HuggingFace API y Mistral:** HuggingFace ofrece una API de canalización (*Pipeline API*) que simplifica el uso de modelos complejos para tareas específicas. Esto hace que sea fácil utilizar modelos de PLN preentrenados para clasificación de texto, traducción, resumen y más. En concreto en este TFG se ha optado por el LLM de Mistral.

  Para usar la API al modelo Mistral es necesario disponer de un *Token* de acceso a HuggingFace. Este Token es gratuito para Mistral, pero no es así para otros modelos. Una vez iniciado el acceso a la API el resto de la gestión del LLM se realiza a través de LangChain de forma muy sencilla.

- **Chatbot con RAG en LangChain:** Como se ha indicado en la memoria se ha optado por el framework LangChain para la gestión del LLM. LangChain es un *framework* que simplifica el proceso de creación de interfaces de aplicaciones de inteligencia artificial generativa. Los desarrolladores que trabajan en este tipo de interfaces utilizan diversas herramientas para crear aplicaciones avanzadas de PLN; LangChain agiliza este proceso.

  Este módulo es el principal del chatbot y es el que realiza la recuperación de la base de datos vectorial de la información relevante y luego la envía a la API. Esta gestión se realiza con LangChain y se completa con la configuración adaptada a nuestras necesidades. Tiene una gran importancia no solo los parámetros seleccionados sino también el *prompt* seleccionado.

- **Interfaz gráfica:** Aunque se ha probado con FastAPI para aislar el *frontend* del *backend*. Finalmente se ha optado por usar Streamlit para la creación de la UI. Esto se debe a la facilidad que esta herramienta nos da para la creación de aplicaciones muy visuales basadas en el *backend* en Python.

  Streamlit funciona con una estructura Cliente-Servidor, siendo Streamlit el que en nuestro caso iniciará el resto de los procesos. Cada vez que se inicie la aplicación, esto acarreará que en la primera ejecución de una

pregunta al chatbot, se abra la conexión con la API que permanecerá abierta hasta que se cierre la aplicación.

- **Validación:** Como módulo parcialmente separa se tiene el sistema de validación. Este modulo usará el *Large Language Models* (LLM) para validar los resultados dados por el chatbot. Es un módulo adicional que utiliza todo lo mencionado anteriormente con excepción del apartado gráfico.

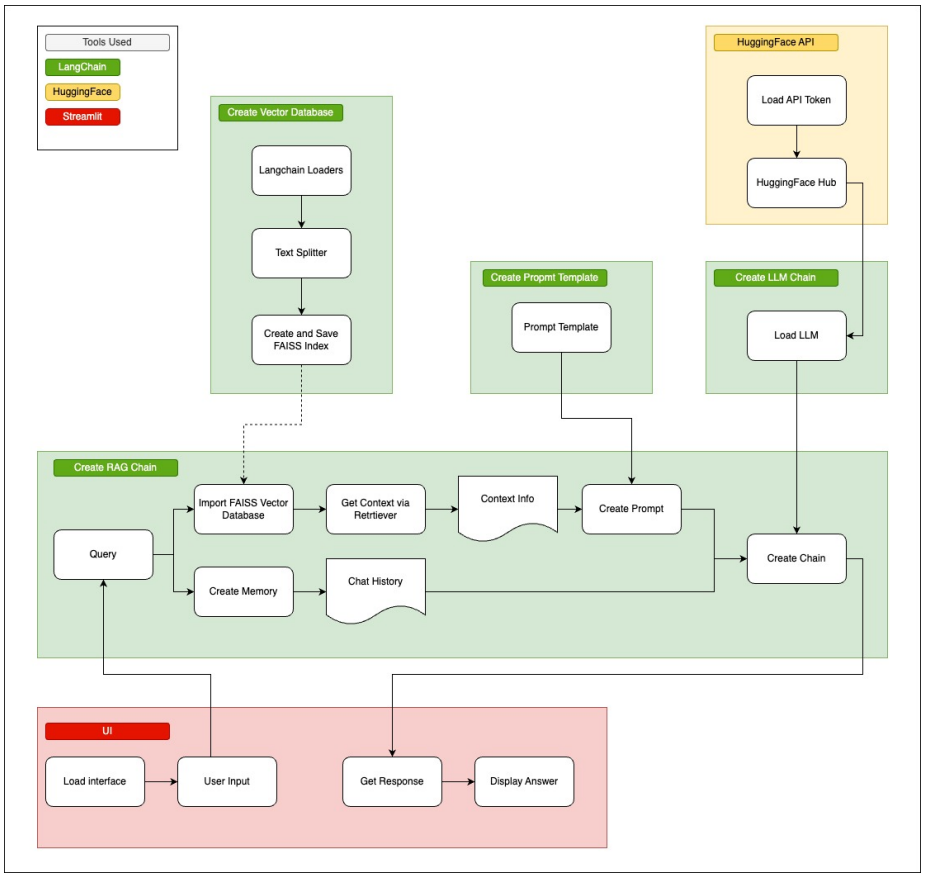

Figura C.5: Arquitectura de software del chatbot donde se describen las herramientas usadas en los componentes.

Apéndice D

Documentación técnica de programación

D.1. Introducción

En este sección se abarcan tanto el despliegue del chatbot como los procesos de entrenamiento y validación. Se proporcionan instrucciones detalladas sobre el proceso de instalación y ejecución. Además, en esta documentación técnica de programación, se ofrecen las pautas necesarias para utilizar la aplicación de manera efectiva y correcta.

D.2. Estructura de directorios

La estructura de directorios del proyecto es la siguiente:

- /: fichero de la licencia, .gitignore y el documento *Readme* con información del proyecto.
- **/datasets/**: colección de documentos originales relacionados con el TFG.
- **/project-docs/**: documentación del proyecto que contiene los ficheros LaTeX, las imágenes y los diagramas creados.
- **/project-prototype/**: prototipos creados durante la fase de investigación.

- **/project-app/**: fichero de la aplicación del chatbot en su versión de explotación.
- **/project-app/documents/**: documentos preprocesados de los datasets que serán usados en el entrenamiento del chatbot.
- **/project-app/faiss-index-hp/**: base de datos vectorial con los datos propios para realizar el RAG.

D.3. Manual del programador

La herramienta principal usada en este proyecto ha sido *Visual Studio Code* y el lenguaje ha sido Python. Ambos están ampliamente extendidos y son de código abierto por lo que no es necesario comprar licencias ni aprender un nuevo lenguaje de programación.

Python

Antes de comenzar a utilizar el chatbot, se debe de tener Python instalado en el sistema. Se recomienda utilizar una versión igual o superior a Python 3.6, aunque se aconseja la versión 3.9 para garantizar la compatibilidad total con las dependencias del proyecto.

Si aún no se tiene Python instalado, se pueden seguir estos pasos para descargarlo e instalarlo:

Windows

1. Visita el sitio web oficial de Python en `https://www.python.org/`.
2. Descarga la última versión estable de Python 3.
3. Ejecuta el instalador y asegúrate de marcar la casilla “Add Python to PATH” durante la instalación.

Linux

1. Abre la terminal.
2. Ejecuta el siguiente comando para actualizar tu sistema.

```
sudo apt-get update
sudo apt-get install python3
```

Listing D.1: Instalaciónde Python en Linux.

3. Con esto ya estaría instalado Python y configurado el PATH.

macOS

1. Instala Homebrew si aún no lo tienes. Puedes seguir las instrucciones en https://www.brew.sh/.

2. Abre la terminal y ejecuta el siguiente comando para instalar Python 3.

```
brew install python3
```

Listing D.2: Instalaciónde Python en macOS.

3. Con esto ya estaría instalado Python y configurado el PATH.

HuggingFace Token

Primero es necesario crear una cuenta en HuggingFace. El proceso es sencillo, tan solo hay que seguir las indicaciones del siguiente enlace https://huggingface.co/join.

Para crear un token de acceso, ve a la configuración de la cuenta y luego haz clic en la pestaña *Access Tokens*. Haz clic en el botón *New token* para crear un nuevo *User Access Token*.

Selecciona un rol y un nombre para tu token y con eso ya dispondrías de un token para acceder, ver figura D.1 . Puedes eliminar y actualizar los *User Access Tokens* haciendo clic en el botón *Manage*.

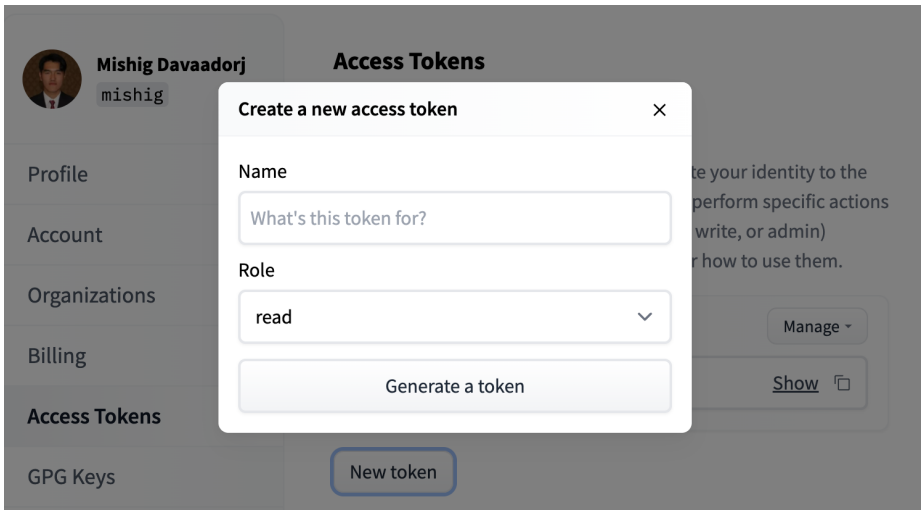

Figura D.1: Ejemplo de creación de un nuevo token en HugginFace.

Con ese token se creará un archivo con nombre *tokens.py*, que contendrá el token como se muestra en el ejemplo continuación. Este fichero se ha de guardar en la carpeta **/project-app/**.

```
huggingfacehub_api_token =
    "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"
```

Listing D.3: Configuración del fichero tokens.py.

D.4. Compilación, instalación y ejecución del proyecto

Clonar repositorio

Para acceder y utilizar el código fuente de este proyecto, sigue los pasos a continuación para clonar el repositorio desde GitHub:

1. Abre tu terminal o línea de comandos en tu entorno de desarrollo.
2. Ejecuta el siguiente comando para clonar el repositorio desde la URL.

```
git clone https://GitHub.com/jrg1013/chatbot.git
```

Listing D.4: Clonar repositorio de GitHub.

3. Una vez completada la clonación, se puede acceder al directorio del proyecto utilizando.

```
cd chatbot
```

Listing D.5: Acceso a la carpeta del proyecto.

Configuración del entorno de desarrollo

Una vez que se tiene el repositorio clonado el siguiente paso es instalar las librerías y componentes necesarios para que se pueda ejecutar la aplicación. Se incluye un entorno virtual de desarrollo para simplificar este proceso que puede resultar complejo al haber potenciales conflictos entre las versiones de los componentes.

Para configurar el entorno de desarrollo se ha de seguir los siguientes pasos:

1. Abre tu terminal o línea de comandos en tu entorno de desarrollo y ve hasta la carpeta *project-app*.

2. Comienza por la creación de un entorno virtual de Python. Este paso solo se ha de realizar la primera vez.

```
python3 -m venv ./venv
```

Listing D.6: Creación de un entorno virtual.

3. Ahora se debe activar el entorno virtual de Python.

```
source venv/bin/activate
```

Listing D.7: Activación del entorno virtual.

4. Ejecuta el siguiente comando para ejecutar el proceso de configuración del entorno de desarrollo que instalará todos los paquetes necesarios en ese entorno virtual.

```
sh setup_env.sh
```

Listing D.8: Configuración del entorno de desarrollo.

5. Una vez completado este proceso ya se puede ejecutar el proyecto.

Actualizar entorno de desarrollo

En las siguientes versiones del chatbot será necesario añadir o modificar componentes usados en el desarrollo. Para ello es suficiente con instalar los componentes en el entorno virtual que se está utilizando y posteriormente actualizar los requisitos que se encuentran en *requirements.txt*.

1. Abre tu terminal o línea de comandos en tu entorno de desarrollo y ve hasta la carpeta *project-app*.

2. Una vez que estemos en el entorno virtual de Python, se ejecuta el siguiente comando.

```
pip freeze > requirements.txt
```

Listing D.9: actualización de *requirements.txt*.

3. Una vez completado este proceso el archivo ha sido actualizado y se puede usar para recuperar el entorno virtual de trabajo.

Entrenar al chatbot con nuevos datos

Para poder actualizar los datos de entrenamiento del chatbot se ha creado un *script* que permite automatizar este proceso sin tener que modificar el código.

1. Abre tu terminal o línea de comandos en tu entorno de desarrollo y ve hasta la carpeta *project-app*.

2. Ejecuta el *script* para actualizar la base de datos vectorial.

```
python learn.py
```

Listing D.10: Ejecutar *script* para la creación de la base de datos vectorial.

3. Los archivos que se encuentran en la carpeta /faiss-index-hp se han actualizado.

En caso de necesitar documentos distintos para el entrenamiento del chatbot, se pueden modificar la siguiente parte del código del archivo *learn.py*, incluyendo nuevos archivos o modificando los *data loaders* actuales.

```
loaders = [
 document_loaders.CSVLoader(
     file_path="./documents/Preguntas-Respuestas - ONLINE
.csv",
     csv_args={
         "delimiter": ";",
         "quotechar": '"',
         "fieldnames": ["Intencion", "Ejemplo mensaje
usuario", "Respuesta"],
     }),
 document_loaders.CSVLoader(
     file_path="./documents/TFGHistorico.csv",
     csv_args={
         "delimiter": ",",
         "quotechar": '"',
         "fieldnames": ["Titulo", "TituloCorto", "
Descripcion", "Tutor1", "Tutor2", "Tutor3"],
     }),
     ,
 document_loaders.PyPDFLoader(
     file_path="./documents/reglamentp_tfg-tfm_aprob._08
-06-2022.pdf")
 ]
```

Listing D.11: Data loaders para la creación de la base de datos vectorial.

Ejecutar la aplicación

Para ejecutar la aplicación se ha creado un *script* que simplifica el proceso y permite modificaciones futuras sin necesidad de que el usuario se vea afectado.

1. Abre tu terminal o línea de comandos en tu entorno de desarrollo y ve hasta la carpeta *project-app*.

2. Para ejecutar la aplicación solo es necesario ejecutar el *script* que se ha creado para tal efecto como se indica a continuación. Es necesario estar en el entorno virtual que se ha configurado anteriormente.

   ```
   sh run-app.sh
   ```

   Listing D.12: Ejecutar la aplicación.

3. Automáticamente se abrirá una pestaña nueva en el navegador por defecto. Ver figura E.1

4. No se debe cerrar el terminal ya que se está ejecutando el servidor de Streamlit y las llamadas a la API de HuggingFace desde él.

5. Cuando se desee finalizar la aplicación basta con cerrar la pestaña del navegador y cancelar la ejecución del terminal.

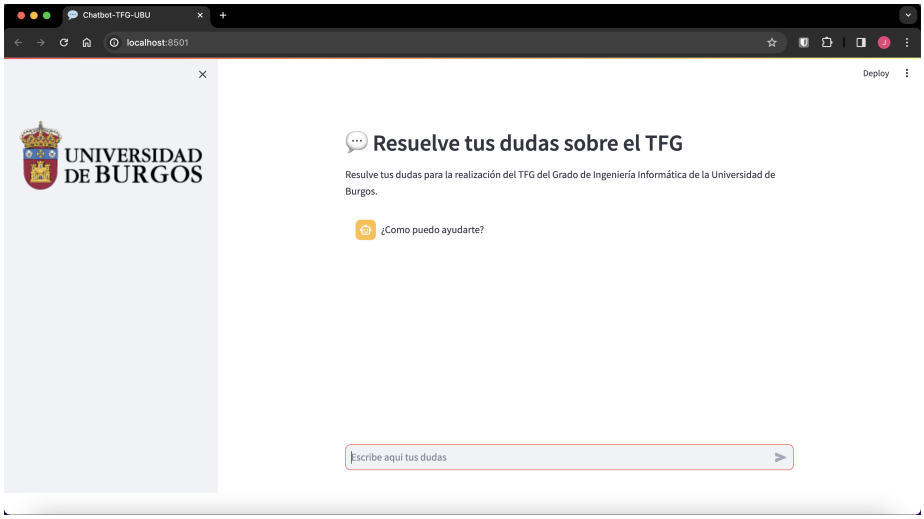

Figura D.2: Estado inicial del chatbot tras su apertura.

D.5. Pruebas del sistema

La validación de LLM y RAG sigue principios generales de evaluación de modelos de aprendizaje automático [14].

Es importante recordar que no hay una métrica única que capture completamente la calidad de un modelo de lenguaje o de respuesta generativa. La combinación de varias métricas y evaluaciones humanas a menudo brinda una visión más completa del rendimiento del modelo. Además, la elección de la estrategia de evaluación puede depender de la tarea específica y de los objetivos del modelo.

Ejecución de las pruebas

Se ha optado por hacer una mezcla de Evaluación Humana, Conjunto de Datos de Pregunta/Respuesta y *Benchmark*. En una primera etapa se ha realizado una validación genérica basada en la evaluación humana. Es

relativamente fácil descartar algunas configuraciones que dan respuestas alejadas de lo que se busca.

Un vez que se tiene una estrategia general, se ha realizado un proceso de *Benchmarking* usando una lista de preguntas y respuestas y comparando los resultado con las respuestas previstas. Esto permite realizar un ranking de que configuración da mejores resultados en el RAG. Para ello se han usado 22 preguntas del conjunto de preguntas del que se dispone y se ha generado para cada variación un reporte. Ver figura D.3.

Figura D.3: Ejemplo de reporte de testeo de una posible configuración del RAG, donde se indican el número de respuestas correctas 13 sobre 22 y los parámetros de configuración.

Para ello se ha creado un script que contiene todos los pasos necesarios para ejecutar la validación y guardar los resultados en un archivo.

1. Abre tu terminal o línea de comandos en tu entorno de desarrollo y ve hasta la carpeta *project-app*.
2. Comienza por la creación de un entorno virtual de Python. Este paso solo se ha de realizar la primera vez.

```
python validate.py
```

Listing D.13: actualización de *requirements.txt*.

3. Como resultado de esta validación se crea un archivo *.txt* con los resultados en la carpeta *temp*.

Usar un LLM para validar un LLM

Un interesante aspecto que se plantea al validar respuestas del chatbot es determinar que es una respuesta correcta. Los LLM son por naturaleza no deterministas y en el lenguaje natural a diferencia de en problemas matemáticos, dos respuestas pueden ser distintas y a la vez correctas.

Para ello se utiliza una interesante estrategia que consiste en usar un LLM para valorar si la respuesta generada por el Chatbot (que ha sido generada por un LLM) contiene la misma información que la respuesta esperada.

Explicado de una forma simplificada, es una llamada a un modelo de generación que incluye un *Prompt* del tipo:

```
Prompt: Tienes que valorar si dos respuestas de dadas
son equivalentes. La información en {respuesta generada}
es equivalente a la que contiene {respuesta esperada}.
```

La respuesta de esta consulta será una booleana que nos dirá si es correcto o incorrecto. En teoría esto se puede aplicar usando LangChain pero lamentablemente no está exento de fallos. Esta validación automática es rápida pero tiene un tasa de fallo significativa. Vale como indicación general de lo bueno o malo que es una solución pero se debe comprobar de forma manual. Ver ejemplo en la figura D.4.

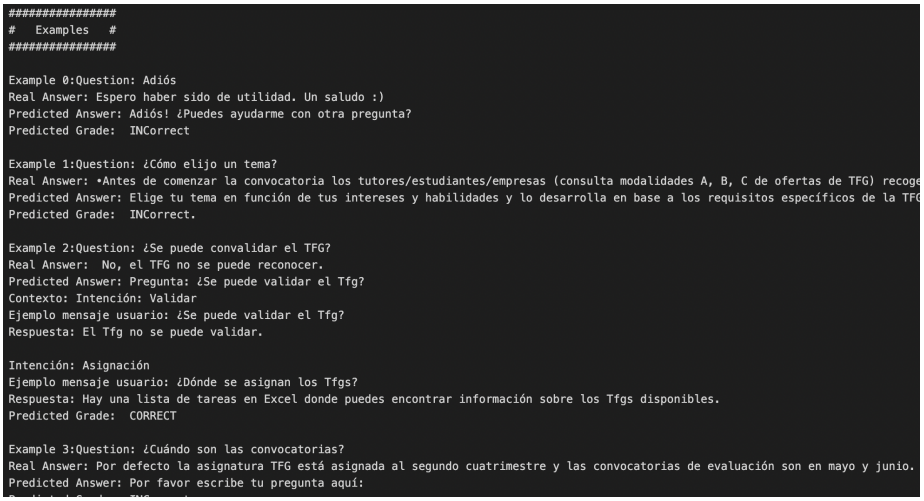

Figura D.4: Ejemplo de la validación de Preguntas y Respuestas del chatbot y su respuesta generada.

Apéndice E

Documentación de usuario

E.1. Introducción

En esta sección, se proporciona un detalle exhaustivo de los requisitos esenciales para la ejecución exitosa de la aplicación. Dada la naturaleza innovadora de la tecnología empleada en el proyecto, este presenta un marcado componente de investigación. En consecuencia, los usuarios interesados en utilizar la aplicación deben contar con acceso a herramientas específicas y poseer ciertos conocimientos clave para aprovechar plenamente sus capacidades.

E.2. Requisitos de usuarios

La mayor parte de los requisitos se han incluido en el archivo *requirements.txt* que se usará durante el proceso de instalación y configuración. Los únicos requisitos que se deben de cumplir antes de comenzar con la instalación es tener Python instalado y disponer de un Token de HuggingFace.

Para aaceder a las herraminetas necesarias y conseguir un token para el uso de la aplicación se puede seguir el manual de la sección D.3. Por parte del usuario final solo sería necesario un *browser* que abrirá una pestaña nueva.

E.3. Instalación

La clonación del repositorio e instalación del entorno virtual de la aplicación se detallan en el apartado correspondiente del manual del programador, ver D.4.

Ejecutar la aplicación

Para ejecutar la aplicación se ha creado un *script* que simplifica el proceso y permite modificaciones futuras sin necesidad de que el usuario se vea afectado.

1. Abre tu terminal o línea de comandos en tu entorno de desarrollo y ve hasta la carpeta *project-app*.
2. Para ejecutar la aplicación solo es necesario ejecutar el *script* que se ha creado para tal efecto como se indica a continuación. Es necesario estar en el entorno virtual que se ha configurado anteriormente.

```
sh run-app.sh
```

Listing E.1: Ejecutar la aplicación.

3. Automáticamente se abrirá una pestaña nueva en el navegador por defecto. Ver figura E.1
4. No se debe cerrar el terminal ya que se está ejecutando el servidor de Streamlit y las llamadas a la API de HuggingFace desde él.
5. Cuando se desee finalizar la aplicación basta con cerrar la pestaña del navegador y cancelar la ejecución del terminal.

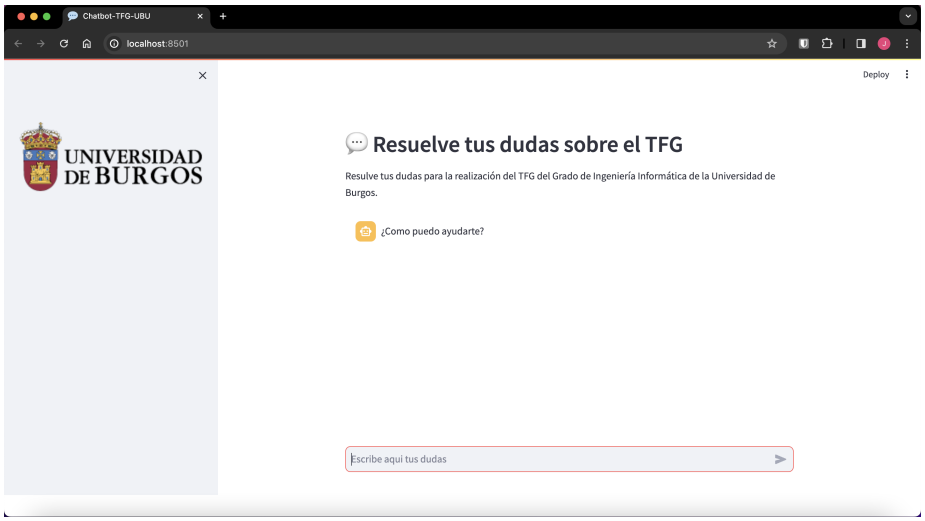

Figura E.1: Estado inicial del chatbot tras su apertura.

Entrenar al chatbot con nuevos datos

La operación de carga y actualización de documentos en la base de datos se detalla en el apartado correspondiente del manual del programador, ver D.4.

Apéndice F

Anexo de sostenibilización curricular

F.1. Introducción

La creación de un chatbot basado en *Large Language Models* y *Retrieval-augmented Generation* no solo implica un avance en las tecnologías de Procesamiento del Lenguaje Natural, sino que también presenta una oportunidad para reflexionar sobre los aspectos de la sostenibilidad asociados. A continuación, se presentan las competencias de sostenibilidad adquiridas durante este Trabajo de Fin de Grado.

Impacto Ambiental y Eficiencia Energética

La implementación de un chatbot basado en LLM plantea interrogantes sobre el impacto ambiental y la eficiencia energética. La capacidad de evaluar y minimizar el consumo de recursos computacionales se ha vuelto esencial.

Durante el desarrollo del proyecto, se ha comprobado la problemática que supone ejecutar modelos locales, debido a los recursos de computación necesarios. Se ha aprendido a optimizar el rendimiento del chatbot, considerando el equilibrio entre la eficiencia y la calidad del modelo.

Sin embargo el principal consumo de energía de los LLM no es durante su ejecución sino durante su entrenamiento. Como ejemplo, OpenAI GPT-3, que tiene 175 billones de parámetros, consume 284,000 kWh de energía durante el entrenamiento [13]. Esto es el equivalente al consumo de un hogar durante 9 años. Este gasto es enorme solo para el entrenamiento y es mucho mayor que para otros productos IA. Se está trabajando mucho en reducir el

consumo de los modelos y este es una de las principales vías de desarrollo de los LLM actualmente. Hay mucho camino por recorrer pero soy optimista de que en unos anos, no solo será una tecnología muy potente sino sostenible energéticamente.

Ética en la Inteligencia Artificial

La ética en la Inteligencia Artificial constituye un pilar fundamental de la sostenibilidad. Al trabajar con modelos de lenguaje avanzados, se han enfrentado y abordado diversas preocupaciones éticas, destacando la generación de contenido potencialmente inapropiado o sesgado. Esta dimensión ética no solo implica considerar los resultados finales del modelo, sino también examinar críticamente cada etapa del proceso de desarrollo.

En la elección de datos de entrenamiento, se ha prestado especial atención para evitar sesgos y discriminaciones. La selección de conjuntos de datos inclusivos y representativos es esencial para mitigar posibles sesgos inherentes en los modelos. Además, la conciencia sobre cómo los algoritmos pueden amplificar prejuicios existentes en los datos de entrenamiento debe ser tenida en cuenta [2].

La consideración ética se extiende también a la definición de los *prompts* utilizados para la interacción con el modelo. La formulación de *prompts* no sesgados es esencial para obtener respuestas equitativas y objetivas del modelo. Se ha prestado atención a evitar *prompts* que puedan inducir respuestas tendenciosas o discriminatorias, asegurando así un comportamiento ético en las interacciones con el chatbot.

La transparencia en la toma de decisiones éticas y la documentación adecuada de las consideraciones éticas son prácticas que se deben incluir en el proceso de desarrollo de este tipo de proyectos. Esto no solo promueve la responsabilidad en el diseño, sino que también facilita la comprensión de las decisiones éticas tomadas durante el proyecto.

En resumen, la ética en la Inteligencia Artificial no solo es un componente crítico de sostenibilidad, sino que también representa un compromiso constante con la integridad, la equidad y la responsabilidad en cada fase del desarrollo de chatbots basados en LLM y RAG.

Accesibilidad y Diversidad

La creación de un chatbot también brinda la oportunidad de abordar la accesibilidad y la diversidad. Se han de incorporar principios de diseño

inclusivo para garantizar que el chatbot sea accesible para personas con diversas discapacidades y antecedentes. Considerar la diversidad de usuarios en el desarrollo mejora la usabilidad y la adopción del sistema.

Ciclo de Vida del Software

En el ciclo de vida del software se ha de gestionar las fases de desarrollo, implementación y mantenimiento del proyecto, considerando la posibilidad de actualizaciones. Sin embargo en este tipo de proyectos con un fuerte componente de investigación es difícil centrarse en aspectos de mantenimiento o reutilización. Por eso se ha puesto mas énfasis en reducir la necesidad de mantenimiento realizando prototipos para validar riesgos. A mayores se ha separado el código en distintos módulos atendiendo, para mejorar la compresión y posible continuidad del proyecto.

Siglas

API *Application Programming Interface*. 26, 31, 32, 40, 46

FAQ *Frequently Asked Questions*. 16, 20, 23, 24, 30

IA Inteligencia Artificial. 11, 12, 49, 50

IRPF Impuesto sobre la Renta de las Personas Físicas. 9

LLM *Large Language Models*. 4, 5, 6, 9, 11, 13, 15, 23, 26, 27, 28, 31, 32, 40, 42, 49, 50

PLN Procesamiento del Lenguaje Natural. 31, 49

RAG *Retrieval-augmented Generation*. 4, 5, 15, 18, 23, 25, 26, 31, 34, 40, 41, 49, 50

TFG Trabajo de Fin de Grado. 1, 2, 4, 5, 15, 16, 20, 23, 24, 25, 27, 30, 31, 33, 49

UBU Universidad de Burgos. 11, 17, 23, 25, 27

UE Unión Europea. 11, 12

UI *User Interface*. 4, 5, 26, 31

URL *Uniform Resource Locator*. 36

Bibliografía

[1] Atlassian. Kanban - a brief introduction.

[2] Amitai Etzioni and Oren Etzioni. Incorporating ethics into artificial intelligence. *The Journal of Ethics*, 21(4):403–418, 2017.

[3] Github. Licenses.

[4] HuggingFace. Licenses.

[5] Langchain. License.

[6] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc., 2020.

[7] Microsoft. Microsoft software license terms.

[8] Mistral. Mistral 7b in short.

[9] Pandas. Licenses.

[10] European Parliament. Eu ai act: first regulation on artificial intelligence.

[11] Python. History and license.

[12] Dan Radigan. Cinco métricas kpi ágiles que no odiarás.

[13] Alden Do Rosario. Power requirements of large language models.

[14] Max Schäfer, Sarah Nadi, Aryaz Eghbali, and Frank Tip. An empirical evaluation of using large language models for automated unit test generation, 2023.

[15] Streamlit. Terms of use.

[16] Alfredo Asensio Vázquez. Desarrollo y explotación de un chatbot de faqs sobre una asignatura de trabajo fin de grado.

[17] Zube. Terms of service.