DESARROLLO DE UN CHAT BOT BASADO EN IA GENERATIVA CON INTEGRACIÓN EN SITIO WEB

Mar Rodriguez Cañero

Resumen– El contenido de este artículo pretende documentar todo el proceso y desarrollo seguido para la creación de una aplicación web que proporciona una interfaz al usuario para diseñar sus propios chatbots, pudiéndolos acabar integrando en sus sitios web. La aplicación web implementa un sistema de autenticación que permite a los usuarios identificarse, configurar sus propios chatbots basados en IA generativa, a nivel de funcionalidad y estilos, realizar pruebas en el mismo sitio web y finalmente, a partir de una API key generada, descargarlo en forma de widget y integrarlo en su sitio web.

Palabras clave– Inteligencia Artificial generativa, chatbot, aplicación web, modelos, prompting, fine-tuning, Large Language Models, interfaz de usuario, API, LangChain, Ollama, sección.

Abstract– The content of this article aims to document the entire process and development followed for the creation of a web application that provides a user interface for designing their own chatbots, which can eventually be integrated into their websites. The web application will implement an authentication system that will allow users to identify themselves, configure their own generative AI-based chatbots in terms of functionality and styles, perform tests on the same website, and finally, with a generated API key, download it in the form of a widget and integrate it into their website.

Keywords– Generative Artificial Intelligence, chatbot, web application, models, prompting, fine-tuning, Large Language Models, user interface, API, LangChain, Ollama, section.

1 INTRODUCCIÓN

La inteligencia artificial (IA) no es más que la disciplina donde se estudian y se desarrollan sistemas capaces de realizar tareas humanas, por lo tanto, de la misma forma que los humanos, estos sistemas son capaces de realizar multitud de tareas y abarcar diversos campos. Inicialmente, debemos entender que la IA se divide en diversas ramas, aunque la más mencionada y conocida es el machine learning.

A menudo las personas tendemos a pensar que es lo mismo, pero el machine learning no es más que una rama de la IA que se centra solo en el proceso en el que los sistemas aprenden a realizar las tareas humanas, el concepto de IA engloba muchos más procesos. El machine learning, a su vez, puede tener diferentes ramas, y de la misma forma que es difícil hablar de la IA sin hablar de machine learning, es difícil hablar del machine learning sin hablar del deep learning. El deep learning no es una rama de la IA, sino que es una rama del machine learning. Esta se centra en entender cómo funciona el cerebro humano y tratar de replicar su comportamiento. En este punto, nos podemos preguntar ¿dónde está la diferencia? Y la respuesta a esta pregunta no es tan compleja si pensamos en el caso de un humano. Nuestro cerebro funciona a través de neuronas que mediante impulsos nerviosos se comunican entre sí siguiendo patrones que nos aportan todos los mecanismos cognitivos y que se almacenan de alguna forma en conocimiento. El deep learning consiste en intentar replicar el proceso que siguen las neuronas para transformar los estímulos en conocimiento, es decir, replicar una red neuronal, mientras que el machine learning se dedica a realizar tareas aplicando deep learning, es decir, usando el conocimiento obtenido a raíz del deep learning. En un ejemplo real, el deep learning consistiría en el proceso de aprender a escribir, mientras que el machine learning consiste en el proceso escribir. El machine learning aprovecha datos ya estructurados y etiquetados, mientras que el deep learning elimina esta parte del procesamiento previo. El sistema resultante de la aplicación de machine learning o deep learning, es decir, el sistema compuesto por una red neuronal que es capaz de realizar tareas humanas, a efectos prácticos, lo denominaremos modelo.

• E-mail de contaco: 1527739@uab.cat
• Mención realizada: Tecnologies de la Informació
• Treball tutorizado por: Marc Ortega Gil
• Curs 2023/24

De la misma forma que el deep learning es una rama del machine learning, la IA generativa también es una rama del machine learning. La IA generativa se basa en crear un modelo que a partir de la recopilación previa de datos (entrada) y su correspondiente entrenamiento, sea capaz de generar nuevo contenido como texto escrito, imágenes, audio o videos. En este proyecto, solo se tendrá en cuenta el texto escrito como entrada, ya que desde el inicio, se está buscando llevar el tema de la IA al desarrollo de un chatbot. También podemos pensar en los NLP (Natural Language Processing) que es la rama de IA que se basa en procesar y comprender el lenguaje humano, pero nosotros también queremos que el chatbot sea capaz de generar respuestas. A este punto es donde se quería llegar y por el motivo por el que se han explicado todos los conceptos anteriores, a los LLM (Large Language Model). Estos son modelos de IA generativa que surgen de la evolución de los NLP, es decir, tienen la capacidad de procesar estos datos de la misma manera que los NLP, pero también tienen la capacidad para generar respuestas basándose en técnicas de deep learning (modelos), es decir, de la misma forma que lo haría un humano.

Como se ha explicado, la IA generativa puede llegar a realizar multitud de tareas y facilitarnos el trabajo a las personas, por ese motivo cada vez está más en auge y las personas hacen más uso de los LLM, surgiendo así la necesidad en la sociedad de aplicaciones capaces de crear LLMs personalizados. De alguna forma, lo que sé en este proyecto es cubrir esta necesidad, desarrollando una interfaz para interactuar con los LLM de la forma más sencilla posible, creando sistemas que realizan la tarea deseada, en definitiva, personalizar sus propios modelos.

2 OBJETIVOS

Después de realizar el proceso de investigación y documentación, ya pudimos empezar y hacernos una idea de los temas que hay que tener en cuenta a la hora de elaborar el chatbot. El objetivo práctico de este proyecto ha sido profundizar en el deep learning, usando de forma práctica redes neuronales y transformers, a través de tecnologías que nos permiten interaccionar con estas, dando como resultado un LLM funcionales personalizados para los usuarios.

Documentar todo el proceso y los pasos seguidos, aparte de la aplicación web, nos ha permitido realizar un artículo resultante que puede servir de guía para introducirse en el mundo de los LLMs y aprender a interactuar de una forma práctica, conociendo todas las posibles alternativas de interacción.

El límite del proyecto ha estado en implementar también un sistema que nos permita conectar el LLM a datos externos como links de los que también pueda aprender. Aplicar Retrieval Augmented Generation (RAG). Conceptos que se desarrollarán posteriormente.

3 ESTADO DEL ARTE

Todos hemos escuchado hablar de la inteligencia artificial y, en la actualidad, lo primero que se nos puede venir a la cabeza a muchos al hablar de la IA es ChatGPT. ChatGPT, aunque nos pueda asustar, nos es más que un ejemplo práctico de dónde puede llegar la IA generativa, un LLM llamado GPT entrenado con millones de datos. Este es el ejemplo más conocido, pero de la misma forma, recientemente han surgido muchos más LLMs como Claude, Bert, Lambda, Lama... ¿Es casualidad que hayan surgido estos modelos durante estos últimos años? La respuesta es que no, no es casualidad. Esto se debe a los avances del deep learning, en concreto, a la red neuronal que usan estos LLMs. Para entenderlo, primero se debe saber que existen muchos tipos de redes neuronales.

Los datos de entrada que se usan para entrenar a estas redes serán millones de párrafos. Estos se pueden dividir en fragmentos más pequeños, como las frases, y a su vez, estas se pueden dividir en fragmentos aún más pequeños, como las palabras. Este proceso se denomina tokenización, se suele hacer por palabras, las cuales serán los tokens. Este paso es común en la mayoría de modelos, la diferencia recae en la manera en la que se procesan cada uno de estos tokens por las distintas redes.

Durante los últimos años, las redes neuronales usadas en este tipo de tareas han sido las redes neuronales recurrentes (RNN), un tipo de red neuronal secuencial. Explicado de forma simple y entendible, estas redes lo que hacen es que coger la primera palabra y esta es procesada por una neurona que le asignará un valor. Este valor resultante, junto con la siguiente palabra, será el input para la siguiente neurona, la cual asignará otro valor a la segunda palabra, de esta manera, este valor para la segunda palabra dependerá del valor de la primera palabra. Así sucesivamente, con todas las palabras, de manera que se establece una lógica entre las palabras, la red está aprendiendo. Independientemente del número de capas de la red, cuando se han intentado mejorar estas redes, siempre acaban surgiendo dos problemas. Al ser modelos secuenciales, no se puede procesar una palabra sin antes procesar la anterior. Son modelos muy lentos de entrenar, no se aprovecha al máximo la CPU. De todas formas, existe un problema mayor, el hecho de que cuantas más palabras se procesan, las primeras palabras pasan a tener menor importancia en la red. Con este tipo de redes y sus limitaciones, es difícil que pudiéramos llegar a crear modelos tan completos como ChatGPT. El punto de inflexión se produjo en 2017 (Attention Is All You Need), cuando aparecieron las redes denominadas transformers. Estas solucionaban el problema del modelo secuencial, pudiendo procesar todas las palabras simultáneamente, aprovechando de esta forma al máximo la CPU, y que independiente de todas las palabras procesadas, la relación no dependa del orden en el que se procesan.La red neuronal usada en estas LLM son los transformers, en las cuales se centra el proyecto, y el proceso de entrenamiento se denomina prompt-tuning y fine-tuning, técnicas esenciales para poder ajustar modelos preentrenados a tareas específicas. Prompt-tuning se refiere a la técnica de ajustar los prompts (instrucciones) dados al modelo para mejorar su funcionalidad, mientras que el fine-tuning implica ajustar los parámetros del modelo.

4 TECNOLOGÍAS ACTUALES

Los años posteriores al descubrimiento de los transformers han demostrado que estos ofrecían un potencial mucho mayor, y los LLMs solo han aumentado su popularidad. Su evolución ha sido muy rápida, alcanzando niveles muy altos en la generación del lenguaje natural. Actualmente, tenemos acceso a una gran variedad de LLMs y tecnologías para interactuar con ellos.

GPT: Desarrollada por OpenAI, esta arquitectura ha sido pionera en la creación de LLMs. Los modelos GPT utilizan una arquitectura de transformer y se entrenan con grandes cantidades de texto para generar lenguaje natural de alta calidad.

MISTRAL: Un modelo de lenguaje grande diseñado para abordar tareas complejas de procesamiento del lenguaje natural (NLP). Utiliza técnicas avanzadas de aprendizaje profundo y redes neuronales transformer para comprender y generar texto de manera coherente y natural.

GEMMA: Otro modelo de lenguaje grande que se centra en tareas avanzadas de NLP. Gemma combina eficiencia y precisión en sus aplicaciones industriales y de investigación.

LLAMA (Large Language Model Meta AI): Desarrollado por Meta, LLaMA es un modelo eficiente y escalable que ha demostrado ser muy efectivo en varias tareas de procesamiento del lenguaje natural (NLP).

CLAUDE: Creado por Anthropic, Claude es un LLM avanzado que utiliza técnicas de transformers y se enfoca en la alineación y la seguridad en la IA.

Hugging Face: Una plataforma y comunidad líder en el desarrollo de LLMs. Proporciona una biblioteca de modelos preentrenados y herramientas que permiten a los desarrolladores implementar, ajustar y desplegar modelos de NLP fácilmente.

Ollama: Plataforma diseñada para facilitar el desarrollo y la implementación de modelos de lenguaje grande (LLMs). Esta plataforma ofrece herramientas y recursos que permiten a los desarrolladores entrenar, ajustar y desplegar LLMs de manera más eficiente y efectiva.

LangChain: Una herramienta emergente que proporciona un marco para construir aplicaciones complejas que utilizan LLMs.

PyTorch: Un marco de código abierto para el desarrollo de aplicaciones de aprendizaje profundo. PyTorch se ha convertido en una herramienta fundamental para el entrenamiento y despliegue de modelos de LLM debido a su flexibilidad y eficiencia. Proporciona una interfaz intuitiva y permite el desarrollo de modelos complejos con facilidad, apoyando tanto la investigación como la producción.

5 METODOLOGÍA

La idea ha sido crear una API rest en la cual tengamos el chatbot. La aplicación web actúa como interfaz para que el usuario configure el chatbot, es decir, se comunique con la API de una forma amigable.

5.1 API

Cuando pensamos en implementar IA de forma práctica, es inevitable para un programador pensar en Python como lenguaje de programación. Además, es el lenguaje usado por defecto en este mundo. Debido a la complejidad de elaborar redes y modelos propios, no solo a nivel de dificultad, sino también a nivel de recursos, la metodología que se ha seguido para la API se ha basado en utilizar modelos Open Source ya existentes, como Llama 2, Mistral y Gemma para elaborar nuestros propios modelos gracias al prompt-tuning y fine-tuning hecho por los usuarios para que el modelo actúe como chatbot siguiendo las necesidades del usuario. Se ha usado Hugging Face y Ollama (dependiendo del caso) para descargar los modelos y LangChain para interactuar con ellos.

5.2 Aplicación Web

Como hemos dicho, la aplicación web actúe como interfaz para que los usuarios configuren y utilicen el chatbot. Para el desarrollo de la aplicación web se decidió usar las tecnologías comunes. Se ha usado NodeJS, específicamente el framework Express, junto con MySQL para el backend. HTML, CSS y Vanilla JavaScript para el frontend.

Para hacer el embedding del chat y entregárselo al usuario para integrar en sus páginas web se ha usado también Vanilla JavaScript (JavaScript puro) asegurándonos compatibilidad con un gran número de páginas que se renderizan con JavaScript en el navegador sin depender de ningún framework ni biblioteca.

5.3 Herramientas extras

GitHub se ha usado para almacenar el código de la aplicación web y de la API, asegurando un control de versiones eficiente. Tenemos la API, la aplicación web y la base de datos, cada una con todas sus dependencias, que en este proyecto son bastante por el tema de la IA. Cuando queremos correr localmente toda la aplicación, necesitamos previamente hacer estas instalaciones localmente, instalar lenguajes, frameworks... Un proceso largo que si deseábamos probar de correr la aplicación en otras máquinas necesitábamos repetir en cada una de ellas. Por ese motivo, y para que el proyecto fuera mas completo y se acercará lo máximo posible a un proyecto real colaborativo, se decidió que era una buena práctica dockerizar la aplicación. Dockerizar la aplicación también nos permite en un futuro poder desplegarla en alguna herramienta de hosting que lee directamente los ficheros docker para la aplicación web. Docker empaqueta la aplicación y todas sus dependencias en contenedores para facilitar el despliegue, distribución y monitoreo de las dependencias. Tanto para la API como para la aplicación web, se definieron sus propios Dockerfile (para MySQL no ya existe una imagen) que crean las imágenes correspondientes y con docker-compose se levantan los tres contenedores de una forma automatizada. Los contenedores estan conectados entre ellos formando toda la aplicación.

5.4 Metodología ágil

En cuanto a metodología de desarrollo de software se ha usado la metodología Agile. Al tratarse de un chat bot, se requería estar en constante periodo de prueba con este para comprobar su evolución y funcionalidad. Esta metodología

se adapta perfectamente a las necesidades al tratarse de una metodología que funciona por iteraciones, donde se busca añadir funcionalidades en cada iteración y completar las iteraciones lo antes posible. En cada iteración se ha realizado las fases de planificación, análisis de requisitos, diseño, codificación, pruebas y documentación.

La planificación inicial del trabajo ha sido la siguiente, teniendo en cuenta que cada fila en blanco representa un salto entre iteraciones:

TAULA 1: DIAGRAMA DE GANTT

Planificación	
Sistema de autenticación	Aplicación Web
HTML + CSS básico	Aplicación Web
Chatbot funcional básico	API
API key	Aplicación web
Administrar modelos creados	API/Aplicación Web
HTML + CSS avanzado	Aplicación web
Widget al usuario	Aplicación Web
Actualización infraestructura	Aplicación Web/API
Prompting	API
Fine-tuning links	API
Idioma	API
Fine-tuning	API
Integración completa	API/Aplicación Web

Se trata de desarrollar una aplicación, por lo que es importante que visualmente se tenga claro cuál será la interfaz. En el apéndice tendremos capturas de todas las secciones de la aplicación web y de algún fragmento importante de código, incluyendo el código de la API. Para más información, consultar el GitHub del proyecto o el diagrama de Gantt mas específico con todas las fechas en la documentación:

- **GitHub:** `https://github.com/Markyrodri2000/AplicacionChatbot.git`

6 DESARROLLO

6.1 Iteración 1

6.1.1 Planificación

La planificación de esta primera iteración buscaba tener una aplicación web básica en la que el usuario pueda loguearse y comunicarse con algún tipo de chatbot a partir de una interfaz amigable. El objetivo era obtener una versión inicial básica de la aplicación y con algún tipo de modelo en la API que se dedique a responder. Los objetivos tampoco se basaron en como responde la API, simplemente obtener una implementación básica.

6.1.2 Requisitos

Los requisitos a cumplir en esta iteración fueron que la aplicación web constará de un sistema de autenticación (tanto para iniciar como para cerrar sesión), que tuviera una interfaz básica que le permitiera al usuario conectar con el modelo de la API a través de un chat y desarrollar el modelo básico en la API.

6.1.3 Diseño y desarrollo

API

Se empezó haciendo una búsqueda de las herramientas que existían para poder descargar los modelos e interactuar con ellos mediante LangChain. La herramienta HuggingFace era la que nos permitiría poder descargar los modelos orientados a realizar la función de un chat, los cuales ya vienen configurados para entender el lenguaje natural (NLP) y entrenados con un dataset general que les permite realizar la función de contestar preguntas (chat). Al descargar las bibliotecas necesarias (transformers, accelerate y LangChain), ya nos dimos cuenta de que eran bibliotecas muy pesadas y que tardaban mucho tiempo en descargarse localmente. Pero aun así, se terminaron descargando correctamente. Después de instalarlas ya pudimos descargar los modelos de HuggingFace, que también tardaron un cierto tiempo considerable. En este punto, debíamos hacer la "puesta en marcha" de los modelos con LangChain. De esta forma ya cumplíamos los requisitos de esta iteración. Un chat funcional básico, el cual inicialmente no controlamos su comportamiento, pero sí que responde preguntas. Solo faltaba ejecutarlo y empezar a hacerle las primeras pruebas para ver si los resultados iban siendo los esperados. Todo parecía funcionar correctamente, hasta que vimos un punto de inflexión en el proyecto. Al ejecutar el chat y realizarle una pregunta, los tiempos de espera eran extremadamente elevados, tardando hasta 45 minutos en responder una pregunta. Todo esto, era debido a que el proyecto se está realizando localmente y no disponemos del hardware necesario para que las respuestas se generen en menos tiempo, teníamos problemas con las limitaciones del hardware. Buscamos diferentes soluciones a este problema y la primera que encontramos es acceder a una API que nos proporcione un chat en la nube, de forma que el código ya no se estaría corriendo localmente. API's como together o replicate nos proporcionan esta función. Decidimos decantarnos por together. Después de implementar el acceso a la API en Python, vimos que si le realizábamos preguntas respondía perfectamente en tiempos razonables. El problema de este tipo de API's es que no podemos modificar el modelo ni realizar prompting, fine-tuning, ... Son modelos cerrados que corren en otra máquina y a través de la API solo se te proporciona acceso a que te responda preguntas, no a que lo modifiques. Era una alternativa que nos solucionaba el problema de momento, pero no era viable durante todo el proyecto debido a que el objetivo es poder modificarlo, es decir, "jugar" con el modelo como queramos. La alternativa que nos quedaba era poder tener acceso a una máquina con mayor hardware. Probamos con Google Colab, que es una plataforma que nos permite desarrollar con Python desde el navegador web y que se ejecuta en los servidores de Google. Esta alternativa suele ser la utilizada en este tipo de casos para acelerar los modelos. Por desgracia, esta plataforma también tiene limitaciones de hardware y

tampoco fue suficiente como para que el modelo respondiera en tiempo adecuado. De la misma forma, también pensamos en Amazon Web Services. AWS nos podía proporcionar acceso mediante "cloud" a una máquina con ciertos recursos. Nos encontramos con el mismo problema, de forma gratuita solo se pueden acceder a ciertos servicios y no sabemos si estos van a ser suficientes.
Debido a que ya teníamos una solución temporal (together), decidimos seguir avanzando en el desarrollo de la aplicación web mientras buscábamos una solución viable a este problema.

Aplicación Web

Al entrar en la aplicación de primeras, en esta primera versión, solo nos encontramos con una barra de navegación que tiene un INICIO donde tenemos absolutamente nada y un INICIAR SESIÓN y REGISTRARSE, un sistema de autenticación. Para poder empezar a interactuar con la aplicación web se requiere registrarse e iniciar sesión. Una vez iniciada sesión, la aplicación nos redirige a la página de inicio donde ya podemos ver y empezar a interactuar con la aplicación.

Solo tenemos esta sección de INICIO y la sección de CERRAR SESIÓN con las siguientes funcionalidades básicas:

- **Inicio:**
  - Configuración del chat (título, nombre del bot, colores, etc.).
  - Interfaz del chat funcional en la parte derecha.
  - Opciones para aplicar cambios y restablecer estilos.
- **Cerrar Sesión:**
  - Finaliza la sesión actual del usuario.

6.2 Iteración 2

6.2.1 Planificación

La planificación de esta segunda iteración iba a estar destinada a avanzar en la aplicación web debido al problema de hardware para seguir desarrollando nuestros modelos. El objetivo era dar un salto de nivel en la aplicación a nivel de interfaz de usuario, HTML y CSS y añadir secciones que debería tener nuestra aplicación para controlar las API keys de los usuarios, para guardar y abrir sus chats, teniendo así el usuario una sección donde guardarlos y finalmente, desarrollar una forma de servirle el widget al usuario.

6.2.2 Requisitos

Teniendo un sistema de autenticación y una interfaz para comunicarse con el modelo, es decir, un chat, los requisitos a cumplir en esta segunda aplicación son chats múltiples chats para cada usuario, y permitir que este pueda administrarlos (crear, editar y eliminarlos). Finalmente, también debíamos ofrecer al usuario la opción de pueda probar como quedaria el widget y descargárselo de forma que desde su propia página web pueda seguir comunicándose con la API. Para la descargar se requiere una API key, por lo que la aplicación también debía generarlas y permitirle al usuario su uso.

6.2.3 Diseño y desarrollo

Aplicación Web

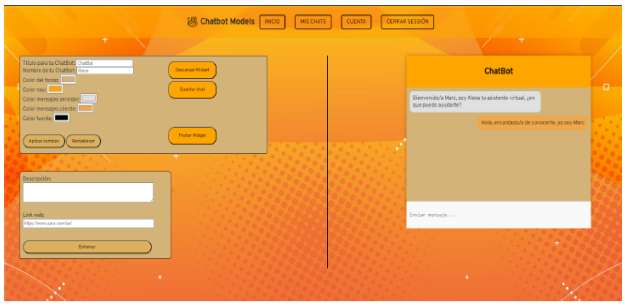

Fig. 1: Interfaz aplicación web: Sección de INICIO.

En esta iteración se han añadido 2 secciones diferentes y diferentes funcionalidades tanto en la sección de INICIO como en las diferentes secciones. La sección de MIS CHATS que es donde podemos acceder a los chats guardados y CUENTA que es donde podemos hacer cambios en nuestra cuenta y acceder a la API keys (implementación funcional en futuras iteraciones). Teniendo ya como resultado una versión provisional de la interfaz que nos permitía definir correctamente las bases del proyecto. Un primer prototipo.

- **Inicio:**
  - Probar o descargar el chat como widget (requiere API key para descargar).
- **Mis Chats:**
  - Acceso a chats guardados.
  - Opciones para eliminar o reabrir chats guardados para edición.
- **Cuenta:**
  - Información de la cuenta.
  - Cambio de contraseña.
  - Generación y gestión de API keys (necesarias para descargar widgets).

Después de casi dos iteraciones completas dedicadas a la aplicación web, estaba prácticamente implementada. Solo faltaba los temas relacionados con los modelos. El sistema resultante consta de 4 secciones:

1. **Inicio:** Se encarga de la configuración del chat. En esta sección, se puede personalizar la interfaz del chat, enviar mensajes que se envían a la API del chatbot y se muestra una indicación de carga mientras se espera la respuesta. También se pueden guardar los chats con diferentes estilos y mensajes.
2. **Mis Chats:** Aquí se pueden acceder a los chats guardados, eliminarlos o abrirlos nuevamente en la sección de Inicio para editarlos o continuar la conversación.

3. **Cuenta:** Permite realizar cambios en la cuenta, como cambiar la contraseña, y acceder a los tokens de API. Se pueden generar, eliminar y copiar tokens para descargar el widget.

4. **Cerrar Sesión:** Opción para cerrar la sesión en la aplicación.

La funcionalidad del widget es la que más problemas nos produjo tanto a nivel de diseño sobre como se iba a implementar esta funcionalidad en la web, tanto a nivel de diseño como a nivel de codificación. Hay que tener en cuenta que el objetivo era únicamente con Vanilla Javascript y no hacer uso de frameworks y librerías. Finalmente, se decidió que el diseño final para el widget fue que desde la sección de INICIO se pudiera probarlo directamente en la página, a partir de un botón que iría cambiando entre habilitar (en el momento que el widget esté desactivado) y deshabilitar el widget (cuando esté activado) o también desde la misma sección de INICIO se pudiera descargar el código necesario para integrarlo en una página web. El widget tendrá la misma apariencia y funcionalidad que el chat configurado. El proceso de descarga del widget requiere ingresar un token de API válido obtenido en la sección de Cuenta. Al pegar el código proporcionado, el usuario obtiene un widget completamente funcional en su sitio web sin problemas de estilos, ya que iframe nos permite incrustar código en otra página, pero sin que entre en conflicto con el código base, es decir, nos permite insertar de una forma sencilla y aislada, elementos de una página web en otra. En este caso, nos permite implantar el widget en la página del usuario.

También se ha implementado un apartado para entrenar el modelo. Este solo se ha implementado de una forma visual, ya que, debido al estado de la API, aún no es posible implementar la funcionalidad.

6.3 Actualización infreaestructura

Debido a las limitaciones de hardware comentadas, inicialmente usamos una API externa (together) para el chatbot, la cual respondía preguntas en tiempos razonables, pero no permitía la interacción directa con el modelo, ya que eran modelos cerrados corriendo en otros servidores. Para solucionar esto, necesitábamos acceso a una máquina virtual con el hardware suficiente para correr nuestros propios modelos y obtener respuestas en tiempos razonables. Evaluamos diferentes servicios en la nube, incluyendo Amazon Web Services, pero finalmente nos decantamos por Azure debido a su especialización en Machine Learning (Azure Machine Learning) y un crédito inicial de 200€ para nuevos usuarios, suficiente para acceder a una máquina virtual con recursos superiores a los de nuestra máquina local o Google Colab.

Una vez iniciada sesión en Azure, creamos un grupo de recursos y un área de trabajo y probamos varias máquinas virtuales. Optamos por una con 16 núcleos, 128 GB de RAM y un disco de 256 GB, a un costo de 1,28€ por hora. Aunque esta máquina no tenía una GPU, era adecuada para nuestras necesidades dentro del presupuesto disponible. No es posible con los créditos de Azure acceder a máquinas con GPU.

Para avanzar en el desarrollo del chatbot, necesitábamos preparar el entorno. Hasta ese momento, estábamos desarrollando usando tres contenedores Docker: uno para la aplicación web, uno para la API y otro para la base de datos. Estos contenedores corrían en local en la misma red. Sin embargo, ahora necesitábamos que la API corriera en la máquina virtual. Teníamos tres opciones: ejecutar el contenedor de la API en la máquina virtual y usar Kubernetes o Docker Swarm, mover toda la infraestructura a la máquina virtual, o conectar la API local con la máquina virtual mediante SSH. Optamos por la conexión SSH, ya que era una solución rápida y eficiente, sin necesidad de realizar grandes cambios en la infraestructura existente, permitiendo que la máquina virtual se dedicara exclusivamente a las tareas relacionadas con los modelos.

La nueva estructura mantenía los tres contenedores locales, pero el contenedor de la API local ahora se conectaba mediante SSH con un contenedor en la máquina virtual. Este contenedor local de la API actuaba solo como puente entre la aplicación y el contenedor de la máquina virtual. De esta manera, podíamos seguir con el desarrollo del chatbot.

De todas formas, después de realizar diferentes pruebas, los tiempos seguían siendo muy lentos. Investigando, encontramos que recientemente se había lanzado Ollama. Ollama es una API que nos permite descargar y correr LLM de una forma local. Al estar realizando diferentes pruebas vimos que los resultados al descargar los modelos desde Ollama eran mucho más rápidos que con HuggingFace y mucho más simple, aparte de que también podemos encontrar una gran variedad de modelos en Ollama, los suficientes para los requerimientos de nuestro proyecto. Decidimos decantarnos por Ollama para descargar los modelos, el cual también tiene compatibilidad con LangChain. Para correr la API de Ollama, creamos un contenedor en la máquina virtual conectado al contenedor de la API. De esta forma, cada vez que la API quiera descargar a un modelo, lo hará desde Ollama que está corriendo en la misma red. Además, al estar corriendo en un contenedor, tiene persistencia (volúmenes), por lo que cada vez que se inicie Ollama no será necesario volver a descargar los modelos ya descargados previamente, estos ya estarán descargados en el contenedor de Ollama. Debido a la facilidad de descargar modelos, decidimos que aparte dé con Llama 2, también íbamos a trabajar con mistral y con Gemma para tener más variedad de modelos y poder compararlos. De todas formas, no decidimos dejar completamente de lado a HuggingFace, ya que en algunos casos nos proporciona modelos menos cerrados y más flexibles, pudiendo, de una forma más sencilla, realizar fine-tuning. Dependiendo de las necesidades usaríamos Ollama o HuggingFace.

Esta es la infraestructura del proyecto desarrollada en el proyecto y, por lo tanto, de la que vamos a hablar de ahora en adelante. El código se encuentra en el GitHub proporcionado anteriormente. En el caso de que se quiera ejecutar toda la aplicación en local, también hay un GitHub habilitado para ello:

- **GitHub:** `https://github.com/Markyrodri2000/`

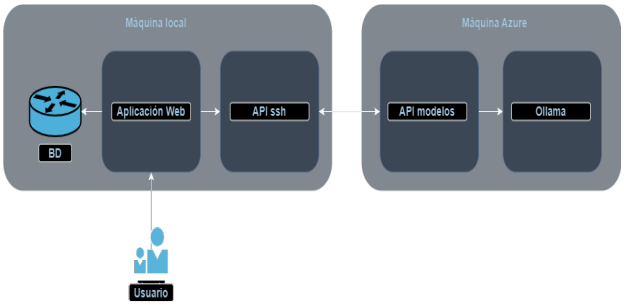

Fig. 2: Infraestructura aplicación completa

`AplicacionChatBotLocal.git`

6.4 Iteración 3

6.4.1 Planificación

Una vez encontrada la solución al problema del hardware, el próximo paso era elaborar un chatbot funcional (con el uso de nuevas tecnologías y substituir a together) que corra localmente en la API modelos, al cual se le pudiera realizar un prompting básico, un fine-tuning a partir de links y al cual le pudiéramos especificar el idioma.

6.4.2 Requisitos

Los requisitos de esta iteración eran claros y concisos, dejar de usar together para crear nuestro propio modelo que corra en la API, capaz de admitir prompts y fine-tuning por parte de los usuarios. Los usuarios también deberán poder seleccionar el idioma con el que quieran que el chatbot responda. Los tiempos de carga y de respuesta del modelo deben ser asequibles para la viabilidad del proyecto que se encuentra en constante período de test y tiempos muy elevados nos complicarían mucho el desarrollo.

6.4.3 Diseño y desarollo

A lo largo del artículo hemos mencionado LangChain, pero no hemos entrado en muchos detalles. En estas secciones, también entraremos detalles acerca de LangChain.

LangChain, como comentamos, es un framework que nos permite crear modelos de una forma sencilla y rápida. Dispone de un gran número de librerías que nos facilita mucho el trabajo. LangChain une diferentes módulos involucrados en el desarrollo de un modelo. En la primera implementación del modelo base, es decir, prompting básico, nos centraremos en el LLM, en el prompt y en las chains. LLM, como sabemos, es el modelo base. Llama 2 lo descargamos a través de Ollama con una temperatura inicial de 0.5. La temperatura es un valor que puede ir de 0 a 1 que indica el nivel de creatividad que queremos que tenga el chatbot. Cuando más cerca de 1 más creatividad. El prompt son las instrucciones que le queremos dar a nuestro LLM, es decir, como tiene que interpretar la entrada y generar la salida. Estas instrucciones se pueden dar a través de plantillas que nos proporciona LangChain (clase PromptTemplate). Inicialmente, esta plantilla le dirá al LLM que actúe como un chatbot funcional básico y le dará como input cada pregunta del usuario. Finalmente, definimos una chain a partir de este LLM y de la prompt dando como resultado un chatbot funcional básico. Las chains lo único que hacen es combinar el LLM con otros componentes. Es el núcleo del flujo de trabajo, en nuestro caso, el chatbot resultante

Pero este chatbot es demasiado básico, ya que no tiene ni memoria, responde cada pregunta como si no le hubieran hecho preguntas anteriormente. Un chatbot funcional básico necesita tener memoria para seguir el hilo y poder mantener una conversación con el usuario. Para ello definimos una memoria a través de la clase ChatMessageHistory, que crea una lista con mensajes de la IA y los mensajes del usuario. Al hacer el prompting, aparte de decirle al LLM que actúe como un chatbot funcional básico, también le pasaremos como contexto el historial de los mensajes, para que responda según el contexto (conversación actual). Ahora sí, teníamos un chatbot general funcional básico que además tiene memoria.

Hasta ahora únicamente necesitamos los módulos LLM, Memory, Prompt y Chain de LangChain y el prompting lo hemos hecho a partir de una descripción y un contexto, pero, ¿qué pasa cuando queremos al contexto de la conversación queremos añadir un link para que al responder las preguntas el chatbot saque información de él? De esta forma, tendríamos un chatbot con memoria capaz de aprender información a partir de links, es decir, el usuario podría crear su propio chatbot dándole la descripción que quisiera y podría pasarle el link de su propia web para que aprenda a responder preguntas acerca de esta, dando como resultado un chatbot personalizado para su propia página web, que era el objetivo del proyecto. Para llevar a cabo este nuevo modelo, necesitaremos añadir también los módulos de LangChain DocumentLoader y VectorStore. Básicamente, lo que hacemos a partir de las herramientas que nos proporciona LangChain es que a partir de un link, lo convertimos a un documento y cargamos la información de este documento en forma de vector usando lo que se les llama “embeddings”. Finalmente, almacenamos este vector en una base de datos vectorial. Ahora, al hacer el prompting, en el contexto, ya le podríamos pasar esta base de datos para que pueda obtener información almacenada en ella, es decir, el link. Hay diferentes tipos de embeddings y bases de datos en LangChain, en nuestro caso hemos escogido el embedding de Ollama y FAISS como base de datos. Al modelo creado le podríamos pasar tanto un link principal y que el modelo aprenda de este link y de todos sus sublinks (secciones) como también le podemos pasar diferentes links para que aprenda de todos ellos de una forma más desestructurada.

Idiomas

En este punto, tenemos un chatbot funcional capaz de darle una descripción de cómo queremos que se comporte, darle un contexto para que siga la conversación y que pueda aprender información acerca de los links, lo único que nos falta es que pueda responder en el **idioma** que deseemos. Para ello, podemos simplemente darle instrucciones al modelo para que responda en el idioma que deseemos, o podemos hacer uso de otras librerías de Python para tra-

ducir cada mensaje. En nuestro caso, después de realizar diferentes pruebas, hemos optado por traducir cada mensaje usando la librería googletrans que es como si fuera un traductor de Google. Hemos optado por esta opción porque de esta manera conservamos que el contexto de la conversación que tiene el modelo siempre sea en inglés, ya que con los datos que han sido entrenados estos modelos son en inglés y de esta manera es más eficiente el modelo.

6.5 Iteración 4

6.5.1 Planificación

En este punto ya teníamos una API capaz de crear modelos a partir de una descripción, del idioma, de una cierta temperatura y de ciertos links, lo único que nos faltaba era adaptar la API para que estos valores los reciba como parámetros y conectarla con la app para que le envíe estos parámetros que deberán ser introducidos por el usuario.

6.5.2 Requisitos

El objetivo era pasar a funciones las implementaciones anteriores, de forma que se recibieran como parámetros el tipo de modelo, que será llama 2, mistral o gemma, el prompt, la temperatura, el idioma y los links, que será un parámetro opcional en el caso de que el usuario quiera añadir uno o más de uno. Con esta implementación podremos hacer peticiones y llamar a cada función según la petición desde la app, ya que la API no devolverá ningún modelo ni valor, pero simplemente con el tipo de petición sabrá lo que tiene que hacer internamente. Con esto tendríamos la API disponible, pero como hemos comentado también deberíamos que adaptar la app para que el usuario pueda interactuar con los modelos y la API ssh que será la que conecte con la API.

Si nos remontamos al la segunda iteración, se ha comentado que en la página de inicio tenía una sección para entrenar el modelo, pero que aún estaba sin implementar. En este punto es cuando debemos implementarla. Inicialmente, la sección de entrenamiento deberá tener los parámetros del modelo básico creado por la API al iniciarse y el usuario debe poder añadir la descripción que queramos al chatbot, el idioma, la temperatura, el modelo y la cantidad de links que quiera. En cuanto a opciones, debería poder tener la opción de entrenar o de restablecer el modelo al inicial.

6.5.3 Diseño y desarollo

Durante el proyecto y, por lo tanto, durante este artículo hemos dividido el desarrollo de la API y el desarrollo de la aplicación web, hablando por separado de cada una de ellas. Debido a que en esta iteración buscamos fusionar las dos partes, no seguiremos diferenciando entre ambas, sino que las trataremos como aplicación comentando cada una de ellas. De la misma forma ocurrirá con las funcionalidades, se fusionarán; serán funcionalidades de la aplicación global, no separadas.

Los cambios realizados en la API han sido los siguientes:

1. Al iniciar la aplicación, se iniciará un diccionario vacío donde se irán colocando las chain cada vez que el usuario cree un chatbot. Esta implementación es debida a que puede haber distintos usuarios que estén usando la app simultáneamente o un usuario con más de un chatbot creado. A la chain creada se añade un chatbot básico para que si el usuario no crea ningún chatbot propio al entrar en la aplicación, este pueda interactuar con este chatbot básico desde el primer momento.

2. Cuando un usuario cree su propio chat en la app, esta realizará una petición a la API para crear sus modelos personalizados a partir de los parámetros introducidos, verificando que si se le pasa un link, el link sea válido para enviar posibles errores. Una vez el usuario lo guardé en la app, este chat será añadido al diccionario de chains. El modelo esté guardado o no, podrá ser modificado por el usuario con un nuevo entrenamiento sin necesidad de reiniciar el historial de mensajes. También tendrá una opción para eliminar el historial de mensajes únicamente, no tiene conexión con el entrenamiento.

3. La API obviamente, también permitirá que un modelo pueda responder a mensajes de un usuario. Una vez tengamos uno o más de un chatbot creado, cuando el usuario quiera interactuar con uno y lo habrá en la app, esta realizará una petición a la API para indicarle el modelo que tiene que responder el mensaje y la memoria (contexto) del chat abierto.

En cuanto a la aplicación web, como hemos definido en los requisitos, se ha modificado la sección de entrenamiento para que inicialmente tenga los parámetros del modelo básico:

Fig. 3: Interfaz aplicación web: Sección de INICIO. Versión final.

Hay que tener en cuenta que en los entrenamientos no se reinicia la memoria del chatbot, es decir, el contexto. El contexto dependerá de los mensajes actuales y se reiniciará una vez se reinicie el chat con la opción restablecer chat.
Anteriormente, también comentamos que la sección de MIS CHATS donde podíamos acceder a los chats guardados. Esta sección ha sido modificada, ya que únicamente podíamos ver el título del chatbot y ahora también queremos que nos muestre información acerca del modelo del chabot. Para ello, hemos añadido un botón para acceder a la información del modelo. Con el uso de localStorage, añadimos una funcionalidad para controlar el chatbot que está abierto en cada momento para indicarle a la API que chatbot debe responder a los mensajes.

En este punto, posteriormente de aplicar los cambios necesarios a la API y a la aplicación web solo faltaba

adaptar la API ssh para conectarlas, es decir, conectar la aplicación web local con la API de la máquina virtual. Al iniciarse toda la aplicación (los contenedores locales) y la API ssh, establecerá conexión ssh con la máquina virtual en la que estará corriendo la API, de forma que podrá ejecutar comando en su terminal. Como la API es una API flask, únicamente con comandos del tipo curl podemos interactuar con ella y además pasarle parámetros.

Fine-tuning

Finalmente, solo faltaba realizar fine-tuning de forma que aparte de links el usuario también pudiera añadir sus propios datasets i, además, realizando fine-tuning podemos usar técnicas como QLoRA para mejorar el entrenamiento y la optimización del modelo. Hasta ahora todos los modelos han sido descargados con Ollama, pero en este caso el modelo debería ser descargado con HuggingFace, ya que como hemos comentado inicialmente, esta herramienta proporciona modelos más libres, flexibles y no tan cerrados, pudiendo realizar fine-tuning. Después de realizar fine-tuning a partir de links, con datasets no debería ser una tarea complicada, pero cuando se trata de un modelo descargado de HuggingFace, debemos tener la CPU y gráfica suficiente para entrenarlos y para usar QLora.

Los resultados nos fueron buenos con la capacidad de computación de la máquina virtual.

Debido a que es un extra de la aplicación, pero no es un requisito imprescindible, decidimos que la mejor opción era no implementar esta funcionalidad para no 'ensuciar' el resultado final. Podemos encontrar el código implementado en la API, pero no es accesible a través de la aplicación web por los usuarios.

7 CONCLUSIONES

La aplicación final desarrollada consta de una API que permite a los usuarios crear y personalizar sus propios chatbots basados en Large Language Model (LLMs). La API ofrece tiene implementada las funcionalidades para realizar prompt-tuning, fine-tuning y la posibilidad de especificar el idioma de respuesta del chatbot, de la misma forma que la aplicación web actúa como interfaz gráfica para que los usuarios puedan tener acceso a estas, interactuar con la API. Finalmente, los usuarios pueden implementar el modelo final, servido por la aplicación a través de cloud, es decir, descargar un widget que pueden integrar en sus propias páginas web. La aplicación web cuenta con las siguientes secciones:

1. **Inicio:** Permite configurar el chatbot (título, nombre, colores, etc.), interactuar con él y guardar los chats.
2. **Mis Chats:** Acceso a los chats guardados, con opciones para editarlos o eliminarlos.
3. **Cuenta:** Gestión de la cuenta del usuario, incluyendo la generación de API keys para descargar el widget.
4. **Cerrar Sesión:** Cierre de sesión del usuario.

En cuanto a la metodología, se ha seguido Agile que tiene un enfoque iterativo de desarrollo, con ciclos de prueba y retroalimentación constantes para mejorar la funcionalidad y el rendimiento de la aplicación. La infraestructura utilizada se ha basado en Azure, aprovechando servicios como Azure Machine Learning para el acceso a máquinas virtuales que permiten el entrenamiento y despliegue básico de los modelos de IA.

7.1 EXPERIENCIA PERSONAL

Dejando a un lado los aspectos técnicos, durante el curso de este proyecto se ha podido descubrir los obstáculos que pueden ir saliendo y aspectos que no se tienen en cuenta en proyectos pequeños, pero que está ahí y realmente son muy importantes a la hora de realizar proyectos reales a más grande escala, ya que conforman las bases.
La importancia de contar con una infraestructura adecuada (hardware y software) para poder entrenar y ejecutar eficientemente modelos de lenguaje grandes acompañado de la complejidad de integrar funcionalidades avanzadas como prompt-tuning, fine-tuning y multilingüismo en un chatbot, siempre teniendo en cuenta la relevancia de ofrecer a los usuarios una interfaz amigable y sencilla, ya que es una aplicación hecha para ellos. Finalmente, la necesidad de considerar aspectos de seguridad y privacidad, como la gestión de API keys.

8 RESULTADOS

Los resultados obtenidos muestran que la aplicación es capaz de generar chatbots personalizados, pero no de una forma eficiente ni con un rendimiento adecuado. Las capacidades son las que son y no es posible obtener los mejores resultados a la altura de LLMs que conocemos y usamos día a día con las capacidades que tenemos a nuestro alcance, totalmente de forma gratuita. El tiempo que se tarda en realizar fine-tuning a partir de un link es elevado, llegando a casi 30 minutos en los casos en los que el link consta de muchas secciones. En el caso de que se introduzcan más de un link de forma desestructurada, los tiempos mejoran considerablemente debido a que no se analizan las subsecciones, llegando a tardar unos 5 minutos d'entrenamiento. En el caso de que no sé realize fine-tuning y simplemente se realice prompting, los resultados son muy buenos; prácticamente no requieren de capacidad de computación y sea el modelo que sea está preparado casi de forma instantánea. En cuanto al chat, tarda aproximadamente 2 minutos en responder preguntas estándares.

De todas formas, la integración con Azure ha permitido escalar la solución y ofrecer una experiencia más fluida a los usuarios.

8.1 VALORACIÓN HERRAMIENTAS

Al inicio del proyecto, nuestro objetivo principal era introducirnos en el campo de la inteligencia artificial generativa a partir de elaborar nuestra aplicación, pero como hemos comentado, un objetivo también era explorar las herramientas disponibles para interactuar con ella. Durante este proyecto, hemos evaluado y utilizado diversas tecnologías que han sido fundamentales para el desarrollo

y el curso del proyecto.

Teniendo en cuenta que este proyecto se trataba de nuestra primera incursión con la IA generativa, el uso de LangChain ha sido fundamental. Es imprescindible para aprender en este campo y considero que está especialmente diseñada y adaptada a usuarios nuevos en este campo y usuarios avanzados. LangChain se complementa perfectamente con Ollama, una plataforma que facilita la interacción con modelos de manera accesible y eficiente. La amplia variedad de funcionalidades de HuggingFace pueden complicar la integración y la interacción con los modelos, tanto si se utiliza LangChain como si no. Ollama ofrece una experiencia más fluida y eficiente, adaptándose perfectamente e incluso mejor que HuggingFace a las capacidades computacionales limitadas. De todas formas, para usuarios con experiencia avanzada y recursos computacionales adecuados, HuggingFace sigue siendo una opción con mucho potencial, ofreciendo mayores posibilidades y funcionalidades. En cuanto a los modelos base usados, Llama2, Mistral y Gemma, hemos encontrado que Llama2 destaca entre el resto. Ofrece versatilidad en las respuestas de una forma muy correcta con tiempos de respuesta y descarga razonables. Por otro lado, aunque Gemma es eficiente, hemos observado que no es ideal para aplicaciones de chatbot debido a sus respuestas menos precisas y no hay tanta diferencia de rendimiento entre Gemma y Llama2 como para que le pase por encima. Mistral es similar a Gemma, pero con tiempos de entrenamiento más largos y tampoco consigue generar mejores respuestas que Llama2.

Este proyecto nos ha permitido no solo explorar nuevas tecnologías en IA generativa, sino también valorar la importancia de seleccionar las herramientas adecuadas según las necesidades específicas del proyecto y las capacidades de cómputo disponibles.

AGRADECIMIENTOS

Quiero expresar mi más sincero agradecimiento a mi tutor por su guía y apoyo a lo largo de este proyecto. Su experiencia, conocimientos y dedicación han sido fundamentales para seguir su curso. Estoy muy agradecido por haberme dado la oportunidad de trabajar en este proyecto bajo su supervisión y espero poder aplicar los conocimientos adquiridos en futuros proyectos.

REFERÈNCIES

[1] Azure Machine Learning. (s. f.). Microsoft.com. Recuperado 19 de junio de 2024, de https://azure.microsoft.com/es-es/products/machine-learning

[2] Diagrama de Gantt simple. (s. f.). Microsoft.com. Recuperado 19 de junio de 2024, de https://create.microsoft.com/es-es/template/diagrama-de-gantt-simple-4bf6b793-490f-4623-84ca-c9c6251a91fc

[3] Express - Node.js web application framework. (s. f.). Expressjs.com. Recuperado 19 de junio de 2024, de https://expressjs.com/

[4] Fine-tuning. (s. f.). LangChain.com. Recuperado 19 de junio de 2024, de https://docs.smith.LangChain.com/old/cookbook/fine-tuning-examples

[5] Generative AI in cyber security. (s. f.). Darktrace.com. Recuperado 19 de junio de 2024, de https://darktrace.com/es/cyber-ai-glossary/generative-ai

[6] Guinness, H. (2023, agosto 16). Meta AI: What is Llama 3 and why does it matter? ZAPIer.com; ZAPIer. https://zAPIer.com/blog/llama-meta/

[7] JavaScript. (s. f.). MDN Web Docs. Recuperado 19 de junio de 2024, de https://developer.mozilla.org/es/docs/Web/JavaScript

[8] LangChain. (s. f.). LangChain.com. Recuperado 19 de junio de 2024, de https://www.LangChain.com/

[9] LLM vs. NLP: 6 key differences and using them together. (2024, enero 10). Kolena. https://www.kolena.com/guides/llm-vs-nlp-6-key-differences-and-using-them-together/

[10] Lutkevich, B. (2024, mayo 14). 19 of the best large language models in 2024. WhatIs; TechTarget. https://www.techtarget.com/whatis/feature/12-of-the-best-large-language-models

[11] No title. (s. f.). Docker.com. Recuperado 19 de junio de 2024, de https://hub.docker.com/

[12] Oliver, N. (s. f.). ¿Qué es la Inteligencia Artificial? ELLIS Alicante. Recuperado 19 de junio de 2024, de https://ellisalicante.org/ia?

[13] Prompts. (s. f.). LangChain.com. Recuperado 19 de junio de 2024, de https://python.LangChain.com/v0.1/docs/modules/model-io/prompts/

[14] ¿Qué es Deep Learning? (2023, mayo 16). Ibm.com. https://www.ibm.com/es-es/topics/deep-learning

[15] ¿Qué es LangChain? (2023, diciembre 26). Ibm.com. https://www.ibm.com/es-es/topics/LangChain
Ratliff, S. (2022, mayo 10). Docker: Accelerated container application development. Docker. https://www.docker.com/

[16] Run JavaScript everywhere. (s. f.). Nodejs.org. Recuperado 19 de junio de 2024, de https://nodejs.org/en

[17] (S. f.-b). Baeldung.com. Recuperado 19 de junio de 2024, de https://www.baeldung.com/cs/rnns-transformers-nlp

[18] (S. f.-c). Datacamp.com. Recuperado 19 de junio de 2024, de https://www.datacamp.com/es/tutorial/fine-tuning-llama-2

APÉNDICE

A.1 Iniciar sesión y Registrarse

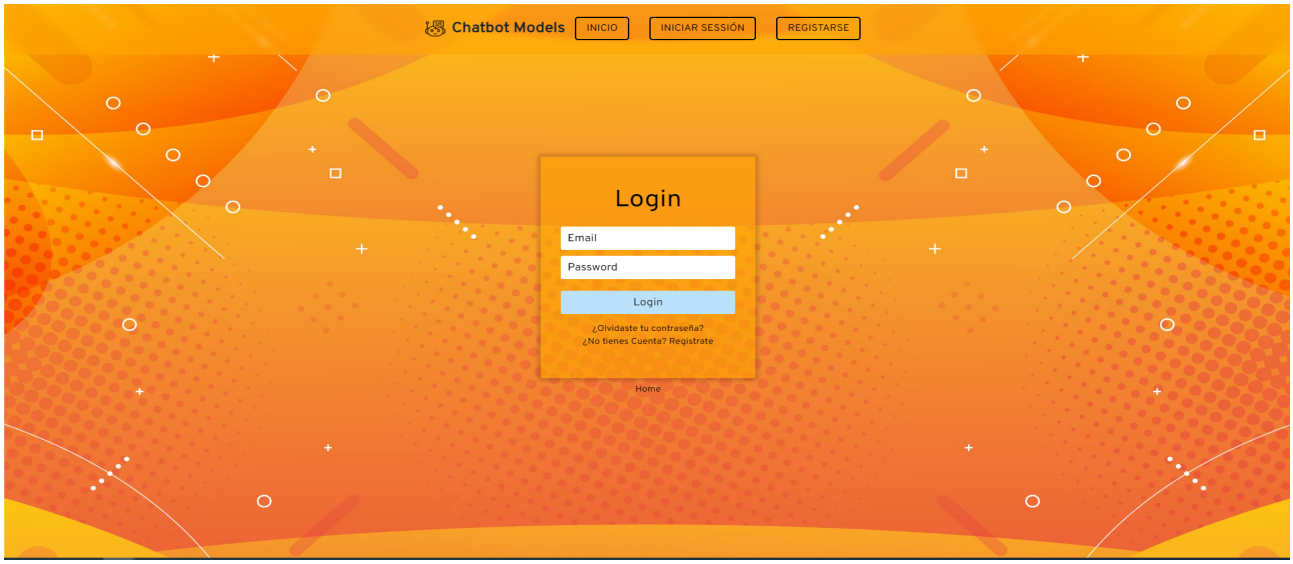

Fig. 4: Interfaz aplicación web: Sección de INICIAR SESIÓN.

A.2 Inicio

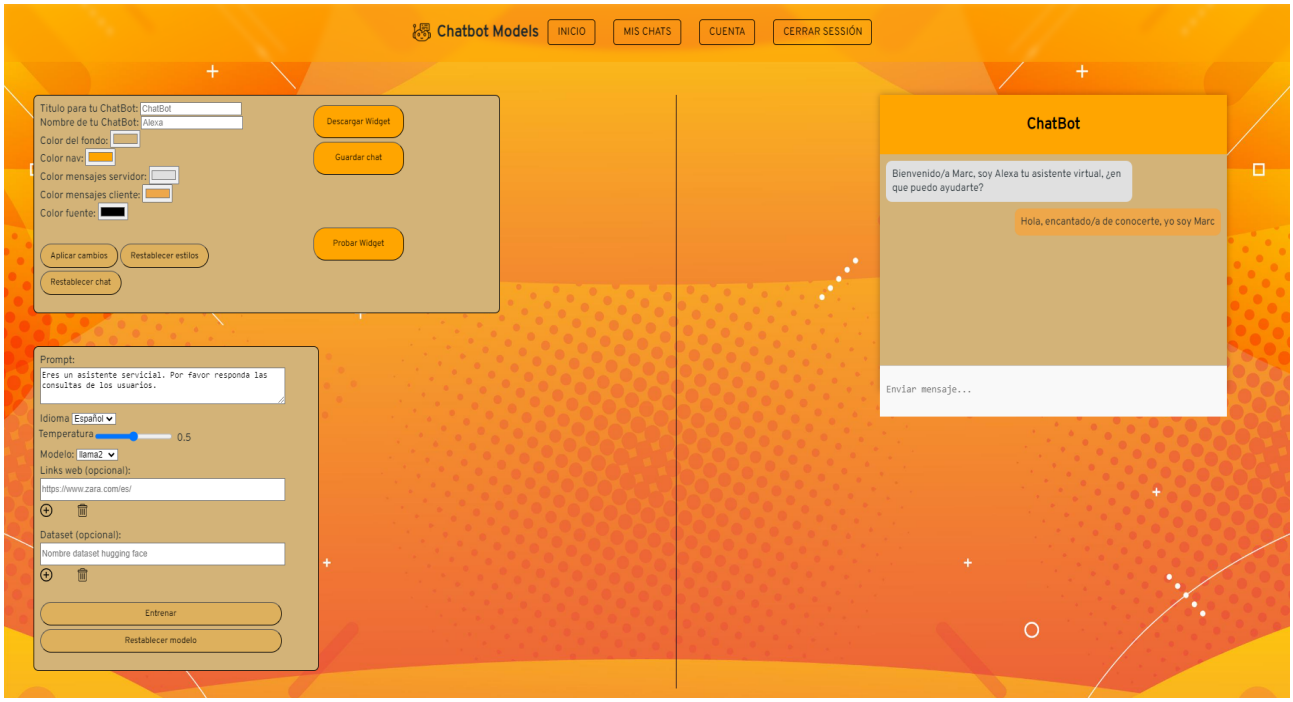

Fig. 5: Interfaz aplicación web: Sección de INICIO.

A.2.1 Funcionalidad estilos y habilitar widget

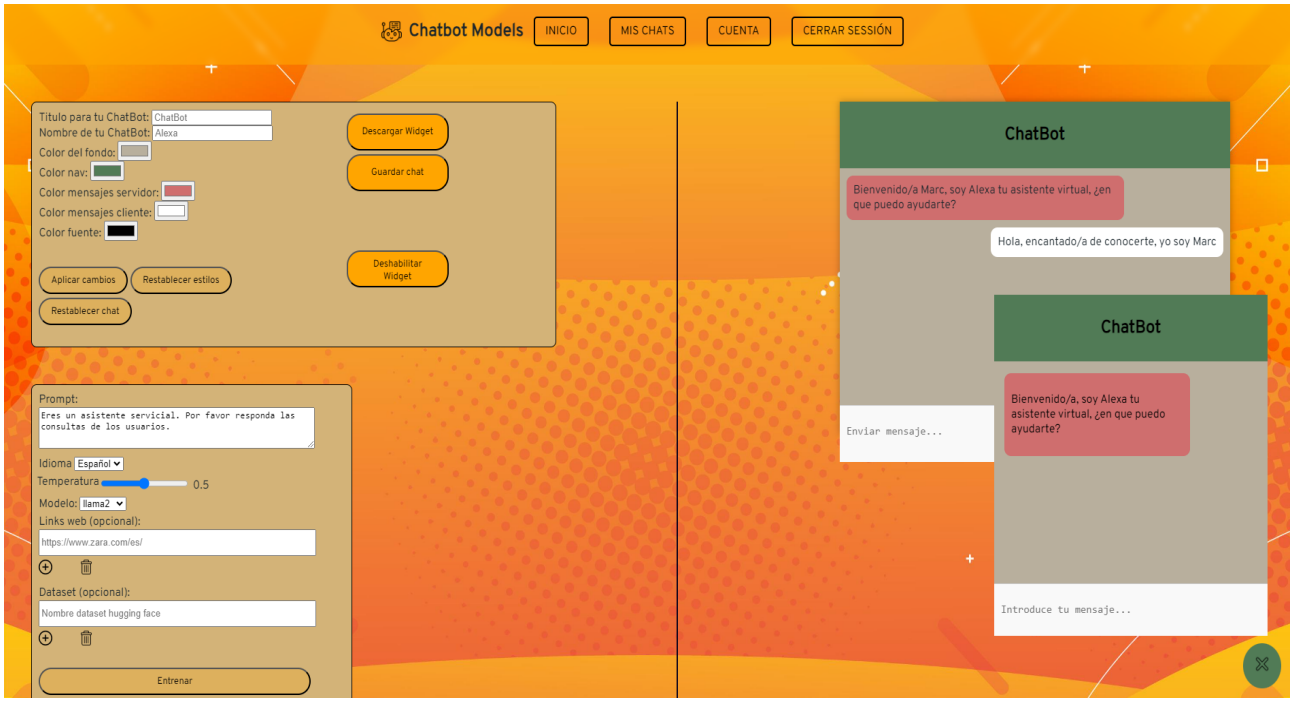

Fig. 6: Interfaz aplicación web: Sección de INICIO. Funcionalidad habilitar widget.

A.2.2 Funcionalidad descargar widget

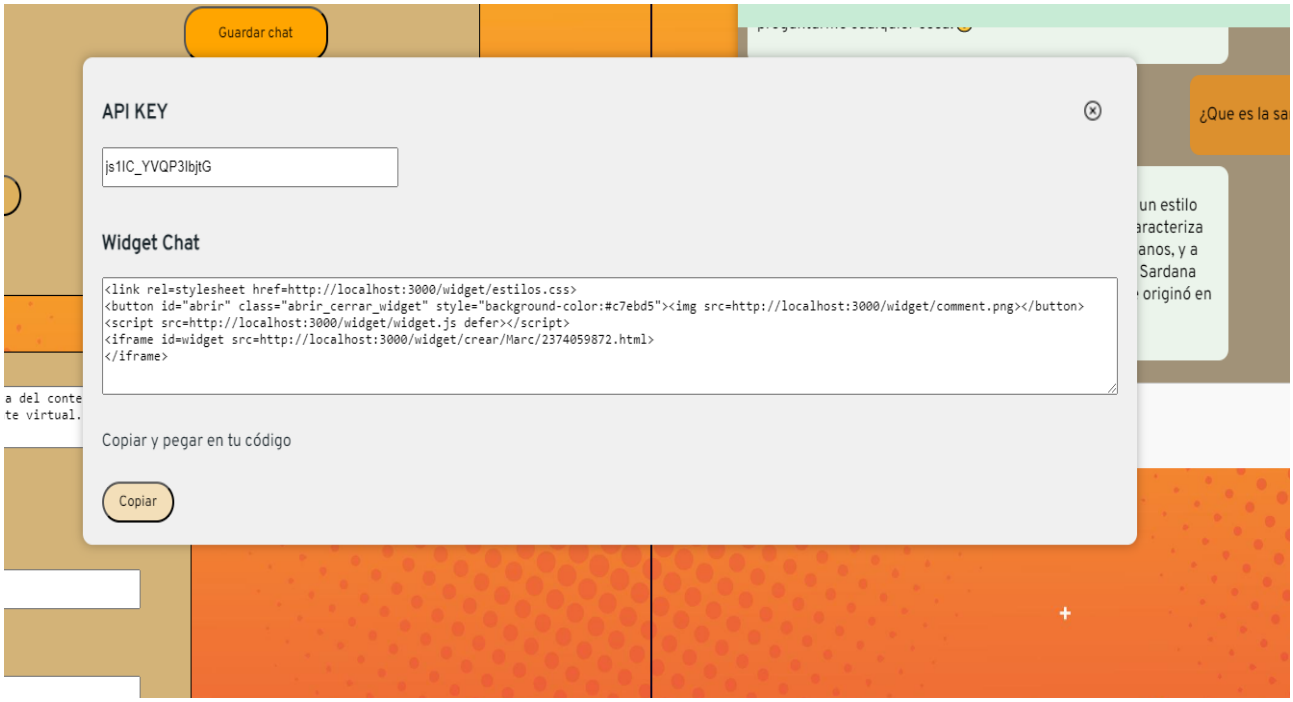

Fig. 7: Interfaz aplicación web: Sección de INICIO. Funcionalidad descargar widget.

A.2.3 Funcionalidad chat prompting

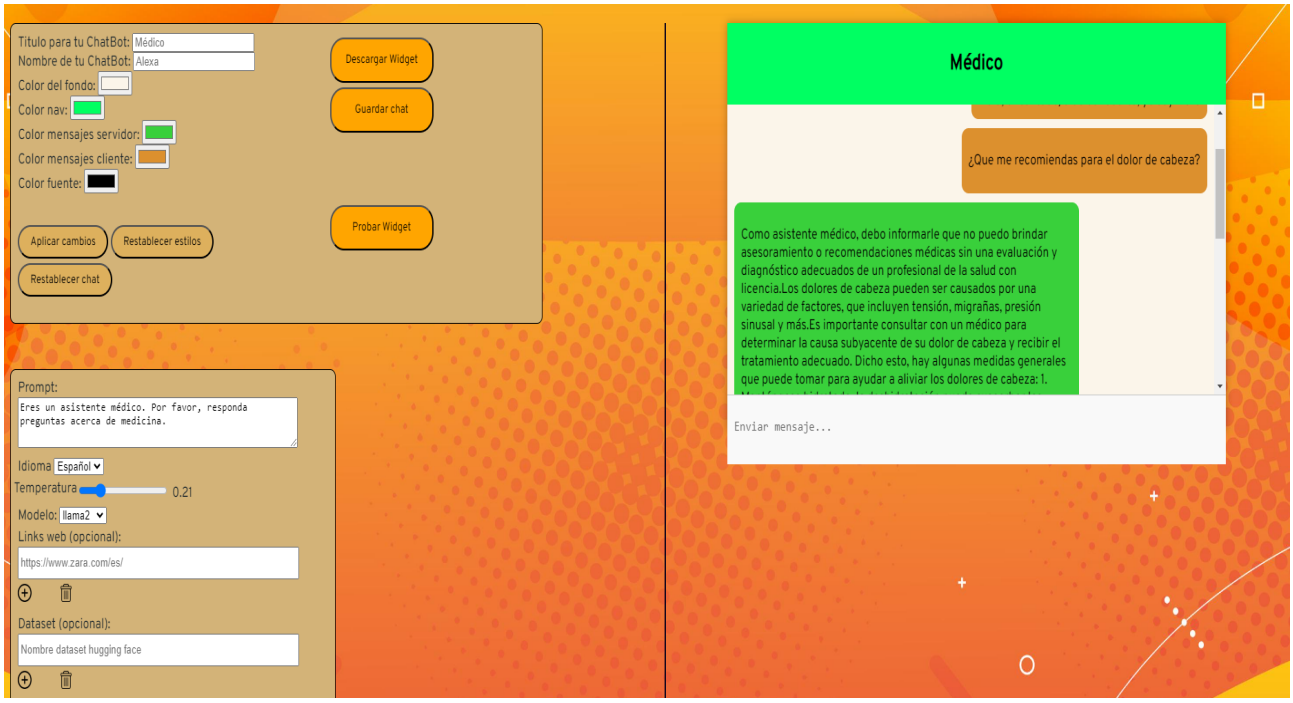

Fig. 8: Interfaz aplicación web: Sección de INICIO. Funcionalidad conversación post entrenamiento.

A.2.4 Funcionalidad chat fine-tuning links

Fig. 9: Interfaz aplicación web: Sección de INICIO. Funcionalidad conversación post fine-tuning link.

A.3 Mis Chats

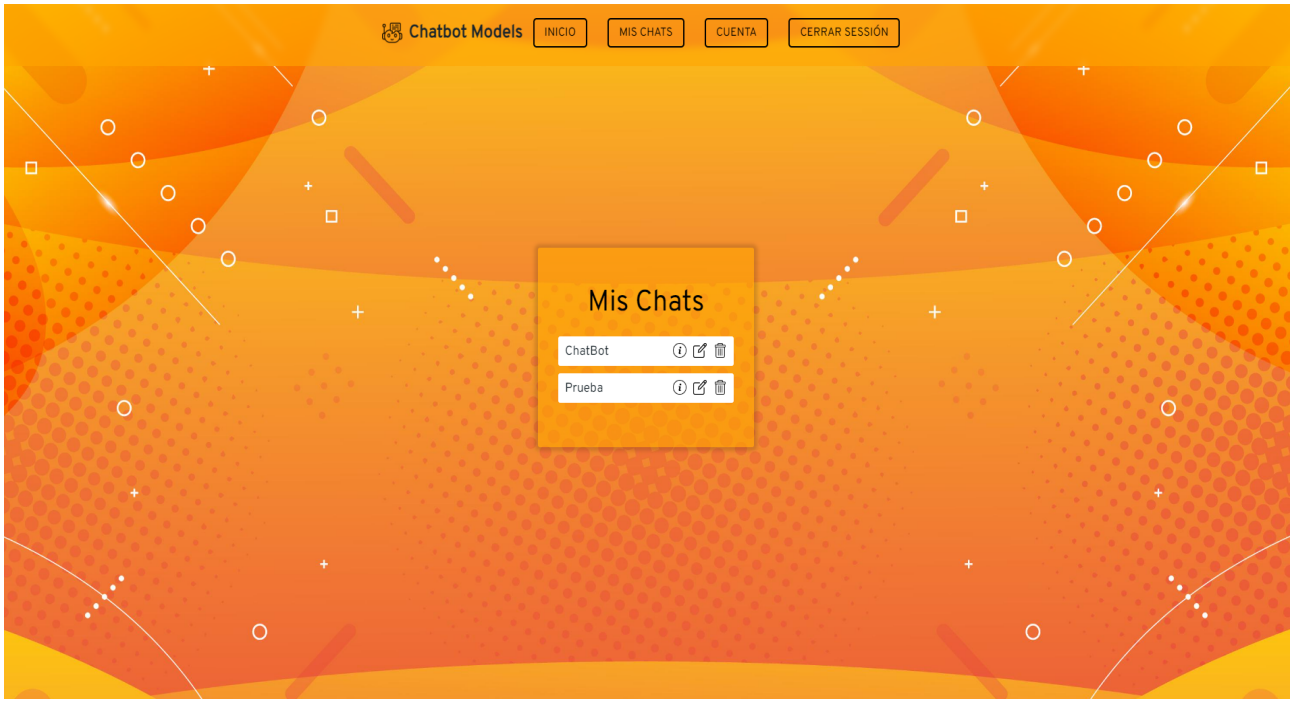

Fig. 10: Interfaz aplicación web: Sección de Mis Chats.

A.4 Web usuario al implementar widget

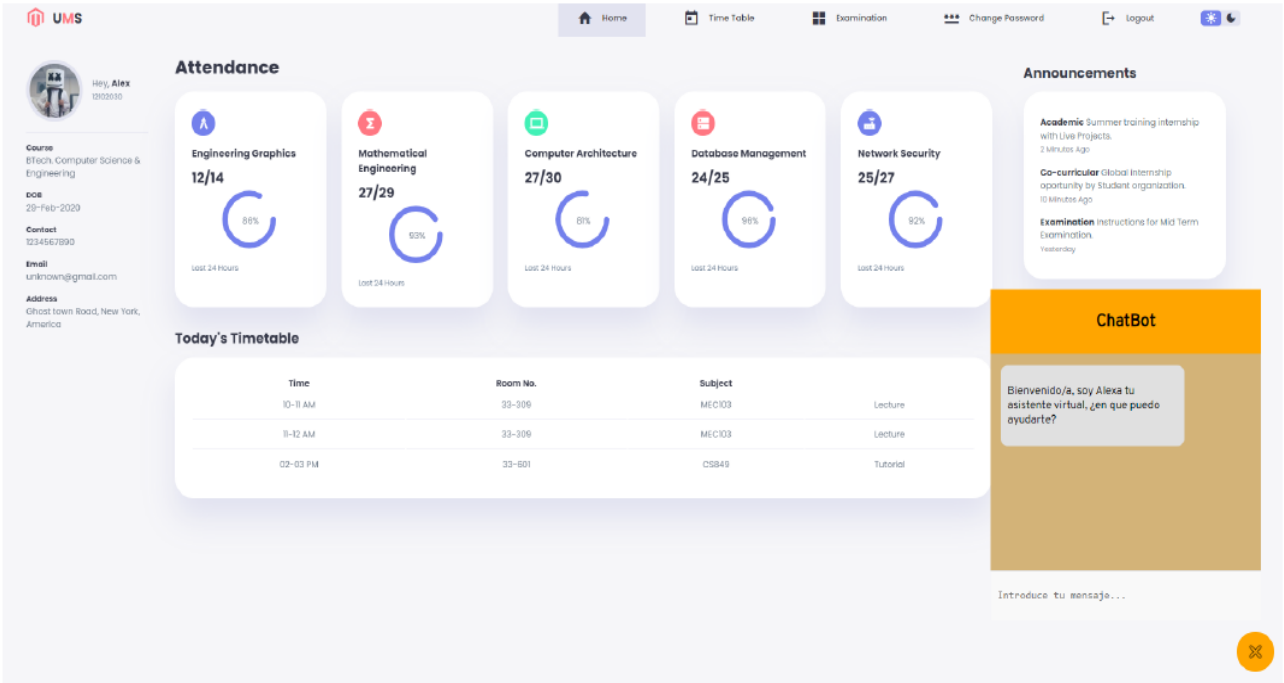

Fig. 11: Widget integrado en aplicación web.