e-ISSN: 1390-5902
CEDAMAZ, Vol. 12, No. 2, pp. 174–182, Julio–Diciembre 2022
DOI: 10.54753/cedamaz.v12i2.1686

Chatbot basado en una versión ligera del modelo BERT para resolver inquietudes relacionadas con matrículas y homologaciones en la Universidad Nacional de Loja

Chatbot based on a light version of the BERT model to resolve concerns related to enrollment and approvals at the Universidad Nacional de Loja

Leonardo Paredes-Rivas [1,*] y Roberth Figueroa-Diaz [1]

[1] *Carrera de Ingeniería en Sistemas/Computación, Universidad Nacional de Loja, Loja, Ecuador*
[*] *Autor para correspondencia: leonardo.v.paredes@unl.edu.ec*

Fecha de recepción del manuscrito: 07/11/2022 Fecha de aceptación del manuscrito: 13/12/2022 Fecha de publicación: 29/12/2022

Resumen—En este artículo se presenta el desarrollo de un chatbot utilizando como red neuronal una versión ligera del modelo BERT denominado DistilBERT, que ayude a estudiantes o profesionales a solventar inquietudes con respecto a matrículas y homologaciones para los estudios de cuarto nivel o posgrados en la Universidad Nacional de Loja (UNL). En este contexto, el proyecto se dividió en dos etapas: en la primera, se hizo una búsqueda bibliográfica en artículos científicos sobre las tecnologías y herramientas compatibles para realizar el ajuste del modelo BERT mediante un entrenamiento en la tarea de preguntas y respuestas; en la segunda etapa, se llevó a cabo el desarrollo del chatbot siguiendo la metodología de Programación Extrema (XP) dividida en cuatro fases: planeación, diseño, codificación y pruebas. En la fase de planeación, se llevó a cabo el entrenamiento del modelo, requisito necesario para la implementación del chatbot. En esta fase se especificaron los parámetros para el entrenamiento modelo y la descripción de forma general del funcionamiento del agente conversacional mediante historias de usuario. En la segunda fase se diseñó la arquitectura, en la que se muestran todos los elementos que formaron parte del chatbot. En la tercera fase se llevó a cabo la programación utilizando los lenguajes de programación Python, JavaScript, Css, Html y el microframework Flask. Finalmente, en la última fase se ejecutaron pruebas de rendimiento, carga y estrés para ver el comportamiento del chatbot al ser sometido a una carga considerable de peticiones.

Palabras clave—Agente conversacional, Inteligencia artificial, Chatbot, BERT, NLP, Metodología XP.

Abstract—This article presents the development of a chatbot using a light version of the BERT model called DistilBERT as a neural network, which helps students or professionals to solve concerns regarding enrollment and homologation for fourth level or postgraduate studies at the National University. of Loja (UNL). In this context, the project was divided into two stages: in the first, a bibliographic search was carried out in scientific articles on compatible technologies and tools to adjust the BERT model through training in the question and answer task; In the second stage, the development of the chatbot was carried out following the Extreme Programming (XP) methodology divided into four phases: planning, design, coding and testing. In the planning phase, the necessary requirements for the implementation of the chatbot were described, where the parameters for model training and the general description of the operation of the conversational agent through user stories were specified. In the second phase, the architecture was designed, in which all the elements that were part of the chatbot are shown. In the third phase, programming was carried out using the programming languages Python, JavaScript, Css, Html and the Flask microframework. Finally, in the last phase, performance, load and stress tests were carried out to see the behavior of the chatbot when subjected to a considerable load of requests.

Keywords—Conversational agent, Artificial intelligence, Chatbot, BERT, NLP, XP methodology.

INTRODUCCIÓN

Hoy en día, el uso de los chatbots o agentes conversacionales está tomando mayor importancia debido a la capacidad que tienen para establecer una comunicación de forma natural y automatizada entre máquina-humano gracias al Procesamiento del Lenguaje Natural (NLP) (Bathija *et al.*, 2020). Muchas empresas están implementándolos en diferentes campos para que los usuarios puedan acceder a la información de manera más rápida y efectiva en cualquier momento del día.

Las instituciones educativas como las universidades, han comenzado a hacer uso de los chatbots para que los estudiantes puedan obtener información sobre diversos temas, como ofertas académicas, requisitos de matriculación, proce-

sos de homologaciones, calendarios académicos, entre otros, que normalmente se suelen obtener de forma presencial en la institución. A través del chatbot los estudiantes pueden hacer preguntas sobre un tema en específico para que este procese la consulta y genere una respuesta en cuestión de segundos (Patel *et al.*, 2019).

Actualmente, existen redes neuronales muy eficientes y entrenadas con el propósito de ser utilizadas en la tarea de preguntas y respuestas y por ende ser acopladas en páginas web o plataformas de mensajería como Telegram, Facebook o WhatsApp para que actúen como agentes conversacionales. Una de estas redes neuronales es el modelo de lenguaje BERT cuyas siglas significan, "Representación de codificadores bidireccionales de trasformadores" (Devlin *et al.*, 2018), donde su arquitectura se basa en una pila de codificadores situados uno encima de otro que forman parte de un Transformer (Vaswani *et al.*, 2017), que procesa como entrada un contexto y una pregunta, que son transformados en vectores numéricos, los cuales pasan por las diferentes capas del modelo hasta llegar a la última, donde se arrojan dos valores que corresponden al inicio y final de la respuesta que se extrae del contexto ingresado al modelo. En este artículo, se propone el desarrollo de un chatbot o agente conversacional, que brinda información sobre preguntas frecuentes relacionadas con los estudios de posgrado en materia de matrículas y homologaciones en la UNL. Para llevar a cabo el proceso de responder preguntas, el chatbot utiliza como red neuronal una versión ligera del modelo de lenguaje BERT, denominado DistilBERT, entrenado previamente en un corpus en español y ajustado con el conjunto de datos SQuAD-v2 traducido al castellano y que posteriormente se lo volvió ajustar con un dataset creado a partir de la información obtenida de los artículos del Reglamento de Régimen Académico de la UNL.

A continuación, se describen las secciones en las que se dividió el presente trabajo. En la primera sección, materiales y métodos, se mencionan todos los recursos utilizados, tanto para la investigación como para el desarrollo del chatbot. En la sección de resultados, se muestran las evidencias que se obtuvieron en el proyecto. En la sección de discusión se hace la interpretación de los resultados obtenidos. Finalmente, en la última sección, se emiten las conclusiones a las que se llegó después de haber culminado el trabajo.

Materiales y Métodos

Búsqueda Bibliográfica

A través de la búsqueda bibliográfica, se pudo realizar una investigación en artículos, libros, revistas y bases de datos científicas confiables, donde se obtuvieron los conocimientos teóricos para el desarrollo del agente conversacional. Gracias a esta técnica se conocieron las tecnologías y herramientas necesarias para ajustar el modelo BERT con un conjunto de datos personalizado, que se creó tomando la información de los artículos del Reglamento de Régimen Académico de la UNL, con respecto a los estudios de posgrado en materia de matrículas y homologaciones.

Metodología XP

Esta metodología permitió desarrollar el chatbot de forma muy flexible a los contratiempos y cambios que se pueden producir en todo el proyecto, la cual permitió realizar de manera rápida el análisis, diseño, desarrollo y las pruebas del chatbot. La metodología usada se dividió en 4 fases:

- Planeación. Fase donde se identificó todo lo necesario para la implementación del agente conversacional.
- Diseño. Etapa que permitió desarrollar una arquitectura general del agente conversacional y la interacción de todos los elementos que van a intervenir.
- Codificación. Fase donde se realizó la programación del chatbot empleando las tecnologías Python, JavaScript, HTML, CSS y Flask.
- Pruebas. Etapa final empleada para realizar las pruebas de rendimiento, carga y estrés al agente conversacional.

Resultados

Tecnologías y herramienta para el ajuste de la red neuronal

En este apartado se llevó a cabo una revisión bibliográfica y para ello se planteó la siguiente pregunta de investigación: *"¿Qué tecnologías y herramientas se pueden usar para el ajuste del modelo BERT en la tarea de preguntas y respuestas?"*.

Los artículos que se obtuvieron para dar respuesta a la pregunta de investigación, se buscaron en bases de datos científicas (IEEE Xplore, ACM Digital Library, Scopus, y arXiv) que, tras pasar por unos criterios de inclusión y exclusión, se seleccionaron aquellos que estaban relacionados con el objeto de estudio. De esta forma los resultados de la revisión se presentan en la Tabla 1.

Desarrollo del chatbot

En el proceso de desarrollo del chatbot se emplearon la metodología XP, la cual se ejecutó mediante un proceso ordenado de 4 fases.

Planeación

En esta fase se especificaron los requisitos para el desarrollo del chatbot, por un lado los parámetros necesarios para el entrenamiento del modelo y por otro la creación de las historias de usuario.

Entrenamiento del Modelo. El modelo escogido para el entrenamiento tiene un peso de 417 MB, denominado "mrm8488/distillbert-base-spanish-wwm-cased-finetuned-spa-squad2-es", el cual se encuentra entrenado previamente en un corpus en español y ajustado con un conjunto de datos llamado SQuAD en su versión 2 también traducido al castellano para la tarea de preguntas y respuestas. Esta red neuronal es un modelo DistilBERT, más rápido, más pequeño y más ligero que la versión original de BERT. Según Sanh et al. (2019), DistilBERT tiene la misma arquitectura de BERT (específicamente BERT_{base} con 110 millones de parámetros y 12 codificadores) (Devlin et al., 2019), de la cual se conserva el 97 % del rendimiento con un 40 % menos de parámetros (66 millones de parámetros y 6 codificadores) y un 60 % más rápido que la versión original.

e-ISSN: 1390-5902
CEDAMAZ, Vol. 12, No. 2, pp. 174–182, Julio–Diciembre 2022
DOI: 10.54753/cedamaz.v12i2.1686

Tabla 1: Resultados de la búsqueda bibliográfica

Código	Titulo	Tecnologías
AC01	BERT-CoQAC: BERT-Based Conversational Question Answering in Context. (Zaib *et al.*, 2021).	TensorFlow Optimizador Adam RAM 16 GB GPU NVIDIA Tesla P100 Biblioteca de Transformers
AC02	Efficient Fine-Tuning of BERT Models on the Edge. (Vucetic *et al.*, 2022).	PyTorch Python Biblioteca de Transformers RAM 16 GB GPU NVIDIA Tesla V100 Volta
AC03	BERT with history answer embedding for conversational question answering (Qu *et al.*, 2019).	Python TensorFlow Optimizador Adam
AC04	Evaluating BERT-based Rewards for Question Generation with Reinforcement Learning (Zhu y Hauff, 2021).	Python PyTorch Biblioteca de Transformers
AC05	Automatic Numerical Question Answering on Table using BERT-GNN (Bagwe y George, 2020).	Python PyTorch
AC06	A Recurrent BERT-based Model for Question Generation (Chan y Fan, 2019).	Python PyTorch Optimizador Adam RAM 24 GB GPU TITAN RTX
AC07	BERT for Conversational Question Answering Systems Using Semantic Similarity Estimation (Al-Besher *et al.*, 2022).	PyTorch Biblioteca de Transformers Optimizador Adam
AC09	Infusing Disease Knowledge Into BERT for Health Question Answering, Medical Inference and Disease Name Recognition (He *et al.*, 2020).	PyTorch Optimizador Adam RAM 16 GB GPU NVIDIA V100
AC10	Pre-training BERT on Resources for Short Answer Grading (Sung *et al.*, 2019).	TensorFlow Python
AC11	Predicting Subjective Features from Questions on QA Websites Using BERT (Annamoradnejad *et al.*, 2020).	TensorFlow Python Biblioteca de Transformers RAM 16 GB CPU 2.00GHz. Intel(R) Xeon(R)
AC12	Questions and Answers on Legal Texts Based on BERT-BiGRU (Zhang y Xing, 2021).	PyTorch Python Biblioteca de Transformers RAM 16 GB GPU GTX-1080Ti
AC13	Two-stage Semantic Answer Type Prediction for Question Answering Using BERT and Class-specificity Rewarding (Nikas *et al.*, 2020).	PyTorch Google Colab RAM 24 GB GPU Nvidia Tesla K80

El modelo escogido se lo entrenó con la ayuda de la herramienta Google Colab, usando una GPU NVIDIA T4 con memoria RAM de 16 GB. El conjunto de datos para el entrenamiento se lo creó, tomando como información los artículos del Reglamento de Régimen Académico de la Universidad Nacional de Loja, el cual está compuesto de 173 artículos, de los que se seleccionaron solo aquellos que estaban relacionados con los temas de matrículas y homologaciones para los estudios de posgrado. En la Tabla 2 se presentan los resultados para la ejecución de varios entrenamientos, donde se han ido modificando los parámetros de entrenamiento (tamaño de lote, épocas y tasa de aprendizaje), para obtener diferentes resultados y en base a ello, mediante una evaluación escoger el modelo con mejor precisión para implementarlo en el desarrollo de un agente conversacional. De esta forma, se puede apreciar que el mejor resultado obtenido fue de 0,8066 (en un rango de 0 a 1) que equivale al 81 % el cual se obtuvo al establecer un tamaño de lote (bach size) de 3, con un número de épocas (epoch) de 7 y una tasa de aprendizaje (lr) de 3e-5.

Tabla 2: Resultados del entrenamiento

Tamaño de lote	Épocas	Tasa de aprendizaje	Pérdida	Precisión	Tiempo de ejecución
4	**5**	2e-5	0,016	0,7857	38 s
8	8	5e-5	0,002	0,6406	56 s
6	7	5e-5	0,005	0,7416	49 s
5	5	4e-5	0,056	0,6888	40 s
5	1	4e-5	1,66	0,6388	8 s
1	8	2e-5	0,026	0,6428	107 s
3	**7**	**3e-5**	**0,007**	**0,8066**	**63** s
1	7	3e-5	0,468	0,6010	91 s
10	10	5e-5	0,145 0,7781	72 s	
20	10	5e-5	0,096	0,7812	70 s
20	19	4e-5	0,004	0,7686	140 s

Historias de usuario. Describen de forma general el funcionamiento del chatbot. Los elementos más destacados que conformaron las historias de usuario son las siguientes:

- Código. Identificador único de la historia de usuario.
- Usuario. Parte beneficiaria que hará uso del sistema.
- Título. Nombre que da una pequeña descripción de la historia de usuario.
- Prioridad. Determina el orden en el que las historias de usuario deben ser implementadas.
- Puntos de estimación. Tiempo estimado para la elaboración del desarrollo de la historia de usuario.

Tabla 3: Historia de usuario 1: Visualización del agente conversacional

Historia de Usuario	**Código**: HU01
Usuario: Usuario Final	
Título: Visualización del agente conversacional	
Prioridad: Alta (Alta/Media/Baja)	**Puntos estimados**: 1
Desarrollador encargado: Leonardo Paredes	
Descripción: Al ingresar a la página web, el Usuario podrá visualizar el agente conversacional, el cual estará disponible en cualquier momento del día.	
Observaciones: Ninguno	

Tabla 4: Historia de usuario 2: Saludo Inicial

Historia de Usuario	**Código**: HU02
Usuario: Usuario Final	
Título: Saludo Inicial	
Prioridad: Alta (Alta/Media/Baja)	**Puntos estimados**: 1
Desarrollador encargado: Leonardo Paredes	
Descripción: El agente conversacional debe realizar un saludo inicial y hacer una presentación donde explique brevemente el tipo de preguntas que se le puede hacer basado en los temas que fue programado.	
Observaciones: Ninguno	

Tabla 5: Historia de usuario 3: Diálogo con el agente conversacional

Historia de Usuario	**Código**: HU03
Usuario: Usuario Final	
Título: Diálogo con el Agente Conversacional	
Prioridad: Alta (Alta/Media/Baja)	**Puntos estimados**: 1
Desarrollador encargado: Leonardo Paredes	
Descripción: El agente conversacional debe realizar una interacción con el usuario, donde debe responder cada una de las preguntas de manera natural.	
Observaciones:El usuario debe escribir la pregunta en la caja de texto del agente conversacional y pulsar el botón de envío. El agente conversacional debe responder en el menor tiempo posible, adicional a ello el chatbot proporcionará el nombre del artículo, donde el usuario podrá consultar la información más detallada de la respuesta. El agente debe notificar al usuario en caso de que no pueda proporcionar una respuesta.	

Tabla 6: Historia de Usuario 4: Diálogo de despedida

Historia de Usuario	**Código**: HU04
Usuario: Usuario Final	
Título: Diálogo de Despedida	
Prioridad: Alta (Alta/Media/Baja)	**Puntos estimados**: 1
Desarrollador encargado: Leonardo Paredes	
Descripción: Para finalizar el diálogo, el usuario puede despedirse del agente conversación, el cual debe responder al mensaje de despedida.	
Observaciones: Ninguno	

Diseño

.

En esta fase, se presenta la arquitectura del agente conversacional, donde se muestra la interacción de todos sus componentes (Figura 1). Los elementos que se pueden observar en la arquitectura son los siguientes:

- El usuario final. Representa cualquier persona, que puede realizar preguntas al bot a través de un navegador con acceso a internet.
- La interfaz gráfica. Es el entorno visual que se puede ver mediante un navegador, por el cual el usuario realizará las preguntas pertinentes al agente conversacional y visualizará las respuestas dadas por el mismo. En el desarrollo de la interfaz se utilizaron las tecnologías de HTML, JavaScript y CSS.
- La interfaz gráfica. Es el entorno visual que se puede ver mediante un navegador, por el cual el usuario realizará las preguntas pertinentes al agente conversacional y visualizará las respuestas dadas por el mismo. En el desarrollo de la interfaz se utilizaron las tecnologías de HTML, JavaScript y CSS.
- Flask. Es un microframework que se usó para montar un servidor web con Python.
- Digital Ocean. Es una plataforma en la nube, donde se pudo desplegar la aplicación web (el agente conversacional) para que diferentes usuarios puedan tener acceso.
- Modelo DistilBERT. Red neuronal que emplearon para la creación del agente conversacional. El modelo se encarga de interpretar las preguntas del usuario y generar una repuesta en base a la misma pregunta y un contexto.
- Base de conocimientos. Se creó basándose en el Reglamento de Régimen Académico de la UNL, a la cual se debe acceder y extraer los contextos que el modelo DistilBERT necesita, para dar una respuesta a las preguntas hechas por el usuario.

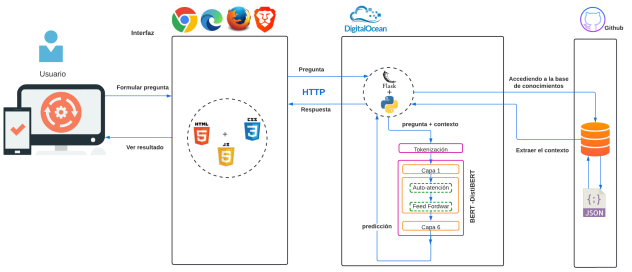

Fig. 1: Arquitectura del agente conversacional

Codificación.

En esta etapa, se utilizaron las tecnologías Flask, Python, JavaScript, Html y Css las cuales se emplearon para la programación del agente conversacional, logrando desplegar la aplicación en un entorno web, para acceder desde cualquier dispositivo. En la Figura 2 se presentan todos los archivos y carpetas que conforman la estructura de la aplicación web (Chatbot).

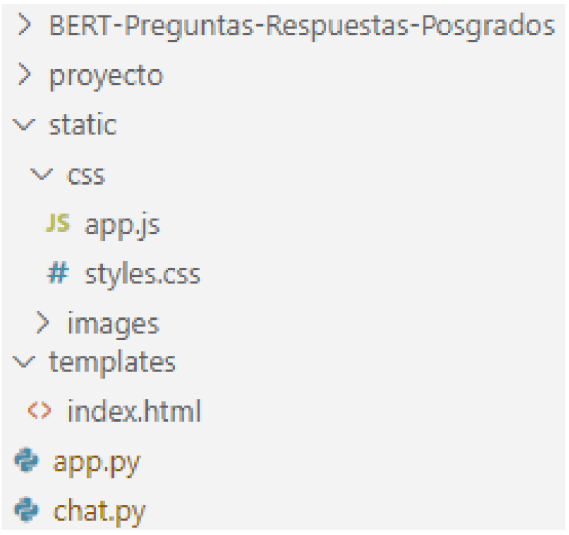

Fig. 2: Estructura del Chatbot

A través del lenguaje de programación Python, se define la forma en como el modelo debe realizar la predicción de la

e-ISSN: 1390-5902
CEDAMAZ, Vol. 12, No. 2, pp. 174–182, Julio–Diciembre 2022
DOI: 10.54753/cedamaz.v12i2.1686

respuesta, al ingresar como entrada a la red neuronal, la pregunta hecha por el usuario y el contexto extraído de la base, conocimientos. En la Figura 3 se observa el código utilizado para que el modelo haga la predicción.

```
#librerías
from transformers import BertForQuestionAnswering, BertTokenizerFast
import torch
import requests
import json
import re
from flask import request
# Importar el modelo
modelo= 'BERT-Preguntas-Respuestas-Posgrados'
#se carga el modelo entrenado para la predicción
model_preentrenado = BertForQuestionAnswering.from_pretrained(modelo)
#transforma las entradas en tokens numéricos
tokenizador = BertTokenizerFast.from_pretrained(modelo)
#establece si se tRabaja con CPU o GPU
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
model_preentrenado = model_preentrenado.to(device)
def get_prediction(contexto, pregunta):
  #se toqueniza las preguntas y el contexto
  entradas = tokenizador.encode_plus(pregunta, contexto, return_tensors='pt').to(device)
  salida = model_preentrenado(**entradas)
  #se calcula el inicio y final de la respuesta  que el modelo predice
  inicio_respuesta = torch.argmax(salida[0])
  final_respuesta = torch.argmax(salida[1]) + 1
  #almacena el inicio-final de la respueta que el modelo predice
  t=entradas['input_ids'][0][inicio_respuesta:final_respuesta]
  respuesta = tokenizador.convert_tokens_to_string(tokenizador.convert_ids_to_tokens(t))
  return respuesta
```

Fig. 3: Predicción de la Respuesta Mediante la Red Neuronal

Para acceder al contexto dentro de la base de conocimiento alojada en GitHub, se hace uso de palabras clave que se extraen de la pregunta realizada por el usuario, las cuales se comparan con las palabras clave ingresadas en la base de conocimientos. La regla que se estableció para acceder a un contexto, es que haya como mínimo dos palabras claves en la pregunta. En la Figura 4 se muestra el proceso para acceder al contexto de la base de conocimientos.

```
url3 = '''https://raw.githubusercontent.com/Leo646/
          TrabajoTitulacion/main/Base_Conocimientos/Base_Conocimientos2'''
resp = requests.get(url3)
datos = json.loads(resp.text)
# accediendo al contexto
for data in enumerate(datos['data']):
  for clave in enumerate(data['palabras_clave']):
    if clave in arreglo:
      t=t+1
      for parrafo in enumerate(data['parrafo']):
        frase=parrafo['context']
        art=parrafo['articulo']
        maximo=t
        maximo1=t
  if maximo==2:
    contexto.append(frase)
    articulo.append(art)
    p.append(maximo)
    maximo=0
  if t==3:
    contexto.append(frase)
    articulo.append(art)
    p.append(t)
  if maximo1>=4:
    contexto.append(frase)
    articulo.append(art)
    p.append(maximo1)
    maximo1=0
  t=0
```

Fig. 4: Accediendo a los Contextos de la Base de Conocimientos

Con en Lenguaje JavaScript se realizaron efectos dinámicos a la interfaz web, como por ejemplo, al pulsar un botón, se debe mostrar o desaparecer el chatbot en la página web. Sin embargo, lo más importante que se hizo con JavaScript, fue rescatar la pregunta que el usurario realiza a través de la caja de texto del agente conversacional, para posteriormente ingresar dicha pregunta al modelo BERT. En la Figura 5 se muestra que, mediante una solicitud POST se rescata la pregunta, la cual se la convierte en una cadena en formato JSON, que sería consumida mediante código Python.

```
//proceso de interacion entre el chatbot, el usuario y la red neuronal
onSendButton(chatbox) {
    //permite acceder a la caja de texto donde el usuario hace la pregunta
    var textField = chatbox.querySelector('input');
    //extrae el valor(pregunta) ingresada por el usuario
    let text1 = textField.value
    if (text1 === "") {
        return;
    }
    //se identifica que la persona que ingrasa la pregunta es el ususario
    let msg1 = { name: "Usuario", message: text1 }
    this.messages.push(msg1);
    //se envia la pregunta con la solicitud POST
    fetch('http://127.0.0.1:5050/predict', {
        method: 'POST',
        body: JSON.stringify({ message: text1 }),
        mode: 'cors',
        headers: {
            'Content-Type': 'application/json'
        },
```

Fig. 5: Obtener la Pregunta del Usuario desde el Chatbot

Finalmente, se desarrolla la página principal donde interviene el microframework Flask el cual se encarga de crear el servicio web para que la aplicación se ejecute y se pueda visualizar (Figura 6).

```
@app.post("/predict")
def predict():
    while True:
        text= request.get_json().get("message").lower()
        respose=get_response(text)
        #el chat respode a una pregunta
        message={"answer":respose}
        return jsonify(message)
#dirección donde se ejecuta la aplicación
if __name__ == '__main__':
    port = int(os.environ.get("PORT", 5050))
    app.run(host='0.0.0.0', port=port)
```

Fig. 6: Ejecución de la Aplicación

Pruebas

Para las pruebas de carga, estrés y rendimiento se usó la herramienta de ApacheJMeter, ya que esta permite medir el desempeño de las aplicaciones con respecto al rendimiento en cuanto a recursos dinámicos y estáticos (Objetos Java, base de datos, lenguajes de programación, protocolos, entre otros). La herramienta permite simular una cantidad de usuarios (grupo de hilos) para comprobar la respuesta del chatbot a una carga simultánea de peticiones. En base a ello, para las pruebas se decidió subir el agente conversacional en la plataforma Digital Ocean, un servicio que permite el alojamiento de aplicaciones en la nube y que ofrece diferentes tipos de recursos, tanto hardware como software dependiendo de las necesidades de los usuarios. En las pruebas se realizaron peticiones HTTP mediante los métodos GET y POST. A través del método GET se realizaron peticiones para recuperar los datos que conforman la interfaz gráfica del agente conversacional, es decir, la estructura, las imágenes y los estilos que forman parte del aplicativo. Con el método POST se llevó a cabo el envío de datos, representando las preguntas que los interesados hacen a través del chatbot. En la Tabla 7 se presentan los datos

ingresados en la herramienta ApacheJMeter para las pruebas de carga y rendimiento mediante el método GET.

Tabla 7: Ingreso de datos para las pruebas de carga en ApacheJMeter mediante el método GET

Número de Hilos	100 usuarios
Tiempo de ejecución de todos los hilos	5 segundos
Número de repeticiones	1 ciclo
Protocolo	http
Dirección de la aplicación web	http://146.190.216.73
Puerto	80
Rutas	/ /static/css/styles.css /static/css/app.js /static/images/logo.png /static/images/burbuja.png /static/images/send.png

La Tabla 8 muestra que la página web fue capaz de soportar 600 peticiones de solicitud-respuesta a través del método GET, en un tiempo de ejecución de 5 segundos, con un rendimiento total de 87,1 segundos, indicando que es capaz de procesar 87,1 peticiones por segundo. Además, se puede observar que la mayoría de peticiones fueron atendidas en un tiempo promedio de 224 milisegundos, siendo el tiempo más pequeño de 15 ms y el más elevado de 4,265 ms. El porcentaje de error fue del 0 % indicando que las peticiones realizadas por los 100 usuarios, de forma simultánea, se ejecutaron en su totalidad sin ningún problema.

Tabla 8: Reporte resumen de las pruebas de carga con el método GET

Etiqueta	#Muestra	Media	Min	Max	%Error	Rendimiento
/	100	220	37	1369	0,00 %	17,8/sec
/static/css/styles.css	100	270	16	4365	0,00 %	15,7/sec
/static/css/app.js	100	197	16	3033	0,00 %	14,4/sec
/static/images/logo.png	100	300	16	3087	0,00 %	13,4/sec
/static/images/burbuja.png	100	187	15	3041	0,00 %	13,7/sec
/static/images/send.png	100	173	17	1354	0,00 %	12,1/sec
Total	600	224	15	4265	0,00 %	87,1/sec

En las pruebas de estrés, se incrementó el número de usuarios y de repeticiones (ver Tabla 9) para determinar los límites del agente conversacional y ver el comportamiento que tiene ante esta situación extrema.

Los resultados presentados en Tabla 10, muestran que el chatbot procesó 4.800 peticiones en 4 iteraciones, con un tiempo promedio de 707 ms. El tiempo mínimo empleado para responder las peticiones fue de 14 ms y el máximo de 63.860 ms. También se tuvo un rendimiento total de 79,3 peticiones por segundo y se puede evidenciar un porcentaje mínimo de error de 0.08 %, señalando que el bot no fue capaz de responder a ciertas peticiones cuando este se sometió a bastante estrés, sin embargo, continuó funcionando sin ningún inconveniente.

Con el método POST, se pudo realizar pruebas en donde se hacían peticiones, simulando una conversación entre el chatbot y el usuario final, para determinar los tiempos que tardaba el agente conversacional en procesar el texto y dar una respuesta. En las pruebas de carga y rendimiento para

Tabla 9: Ingreso de datos para las pruebas de estrés en ApacheJMeter mediante el método GET

Número de Hilos	200 usuarios
Tiempo de ejecución de todos los hilos	5 segundos
Número de repeticiones	4 ciclo
Protocolo	http
Dirección de la aplicación web	http://146.190.216.73
Puerto	80
Rutas	/ /static/css/styles.css /static/css/app.js /static/images/logo.png /static/images/burbuja.png /static/images/send.png

Tabla 10: Reporte resumen de las pruebas de estrés con el método GET

Etiqueta	#Muestra	Media	Min	Max	%Error	Rendimiento
/	800	473	35	15457	0,00 %	11,8/sec
/static/css/styles.css	800	198	16	63860	0,13 %	13,7/sec
/static/css/app.js	800	227	16	13123	0,00 %	14,4/sec
/static/images/logo.png	800	172	15	56218	0,13 %	13,4/sec
/static/images/burbuja.png	800	135	14	62330	0,13 %	13,9/sec
/static/images/send.png	800	159	15	59825	0,13 %	12,1/sec
Total	4800	707	14	63860	0,08 %	79,3/sec

este método se emplearon los mismos datos de la Tabla 7, obteniendo como resultado lo que se observa en la Tabla 11.

Tabla 11: Reporte resumen de las pruebas de carga con el método POST

Etiqueta	#Muestra	Media	Min	Max	%Error	Rendimiento
/predict	100	37798	15553	70346	0,00 %	46,0/min
Total	100	37798	15553	70346	0,00 %	46,0/min

Los resultados indican que el agente conversacional pudo responder las 100 peticiones realizadas por los usuarios, con un tiempo promedio de 37,798 ms. El tiempo mínimo de respuesta fue de 15,553 ms y el máximo de 70,346, con un rendimiento de 46 peticiones por minuto y un porcentaje de error del 0 %. Los tiempos de respuesta de las peticiones son más elevados a través del método POST. Esto se debe a que al momento de que un usuario realiza una pregunta al agente conversacional, el modelo DistilBERT consume bastante memoria y CPU para analizar un texto completo y dar una repuesta en base al mismo, haciendo que los tiempos también se incrementen, sin embargo, a pesar de todo ello, el agente fue capaz de responder a las peticiones sin ningún error, mostrando que puede manejar 100 peticiones simultáneas durante un ciclo. Al incrementar las repeticiones a 4 ciclos generando un estrés en chatbot lo que se observa en la Tabla 12.

Tabla 12: Reporte resumen de las pruebas de estrés con el método POST

Etiqueta	#Muestra	Media	Min	Max	%Error	Rendimiento
/	400	4374	12587	81366	78,00 %	45,0/min
Total	400	4374	12587	81366	78,00 %	45,0/min

De acuerdo a los resultados, se generó un 78 % de errores al atender 400 peticiones de forma concurrente, esto quiere decir que la mayor parte de las solicitudes realizadas por los

e-ISSN: 1390-5902
CEDAMAZ, Vol. 12, No. 2, pp. 174–182, Julio–Diciembre 2022
DOI: 10.54753/cedamaz.v12i2.1686

usuarios no fueron atendidas, ya que el aplicativo estuvo sometido a bastante estrés provocando que el servidor donde se alojaba lleguara a reiniciarse. En las figuras 7 - 10, se puede observar la interfaz gráfica del agente conversacional vista desde un navegado a la cual se puede acceder mediante el siguiente enlace `http://146.190.216.73/`.

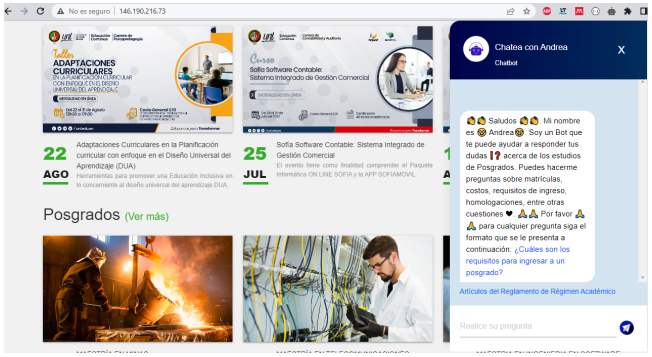

Fig. 7: Presentación del chatbot

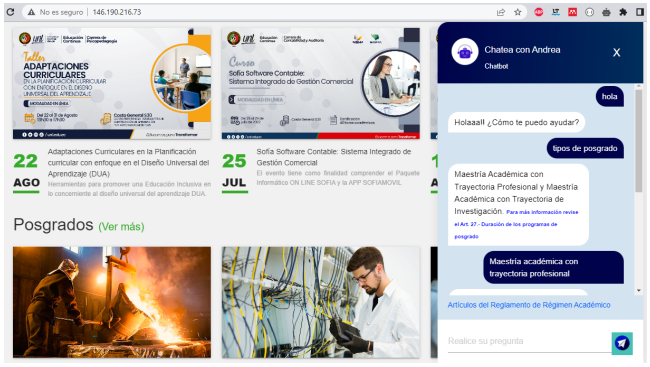

Fig. 8: Interacción con el chatbot. Ejemplo 1

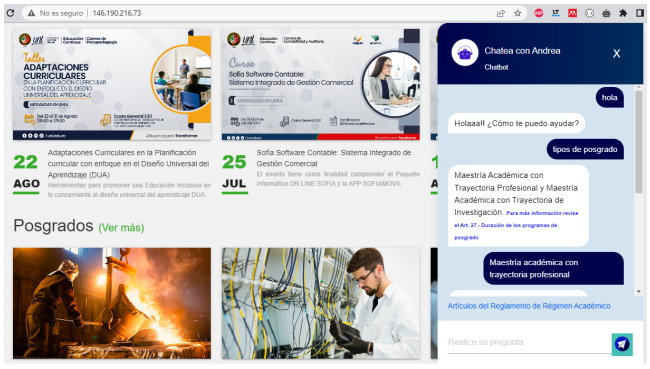

Fig. 9: Interacción con el chatbot. Ejemplo 2

Discusión

La búsqueda bibliográfica permitió conocer las tecnologías y herramientas que se pueden usar para ajustar el modelo de lenguaje BERT, misma que fueron usadas para la versión ligera que se seleccionó para este artículo. A través de la investigación y después de pasar por unos filtros de inclusión y exclusión, se encontraron 13 artículos científicos que dieron información para responder a la pregunta de investigación planteada inicialmente. En la mayoría de los artículos se menciona que el lenguaje de programación para el ajuste del modelo es Python junto con las bibliotecas PyTorch y

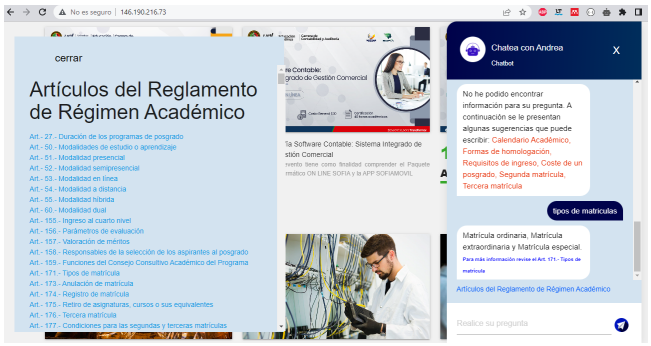

Fig. 10: Interacción con el chatbot. Ejemplo 3

Transformers. Además, como recursos mínimos en cuanto a hardware se han empleado GPUs con memoria RAM de 16 y 24 GB.

Con la información que se obtuvo, se pudo llevar el desarrollo del agente conversacional siguiendo la metodología XP dividida en cuatro fases. Inicialmente, en la primera fase se llevó a cabo el entrenamiento del modelo BERT, donde se escogió una versión ligera denomina DistilBERT, la cual tiene la misma arquitectura que el modelo original, pero es un 60 % más rápido y un 40 % más pequeño. Para el entrenamiento se usó la herramienta Google Colab, la cual tiene a disposición recursos de hardware gratuitos como GPUs que permitieron realizar el entrenamiento mucho más rápido (con intervalos de tiempo de 8 segundos a 140 segundos), además de trabajar con el lenguaje de programación Python. El conjunto de datos para el entrenamiento se lo creo con información del Reglamento del Régimen Académico de la UNL con los temas de matrícula y homologaciones para los estudios de posgrados. El entrenamiento del modelo se lo realizó con una GPU NVIDIA T4 con capacidad de memoria RAM de 16 GB, que fue proporcionado por Google Colab. Se llevaron a cabo 5 ejecuciones de entrenamiento donde se modificaron los parámetros para entrenar el modelo (tamaño del lote, número de épocas y la tasa de aprendizaje), dándoles valores diferentes para obtener diversos resultados y en base a ello, escoger un modelo con una precisión alta. De acuerdo a los resultados, el valor más alto que se consiguió fue de 0,8066 (81 %), dicho valor se lo consiguió al establecer el tamaño de lote en 3, el número de épocas en 7 y la tasa de aprendizaje en 3e-5. Una vez entrenada la red neuronal, se llevó a cabo la creación de 4 historias de usuario donde se describió de forma general el funcionamiento del chatbot. En la segunda fase, se diseñó una arquitectura en la que se muestran todos los elementos que se usaron para la implementación. En la tercera fase se llevó a cabo el proceso de codificación, donde se utilizaron los lenguajes de programación, Python, para codificar la forma en como el modelo debe realizar la predicción de las respuestas en base a las preguntas realizadas por los usuarios; HTML, para construir la estructura de la interfaz gráfica del agente conversacional, CSS, para dar estilos visuales; JavaScrript, para generar efectos y rescatar las preguntas generadas por los interesados a través de la interfaz gráfica del bot; y Flask para levantar un servicio web y ejecutar la aplicación.

Como paso final se realizaron las pruebas de rendimiento, carga y estrés, utilizado la herramienta ApacheJMeter. A través de esta herramienta se hicieron simulaciones, de peti-

ciones http mediante los métodos GET y POST. Con el método GET se observó que el agente conversacional fue capaz de soportar 100 usuarios de forma simultánea, atendiendo un total de 600 peticiones sin ningún tipo de error durante una sola ejecución y tomándole un tiempo promedio, para atender la mayoría de peticiones de 224 ms con un rendimiento de 81,1 solicitudes por segundo. Sin embargo, al aumentar la cantidad de usuarios a 200 y el número de repeticiones a 4 ciclos, atendiendo en total 4.800 peticiones, se evidenció un error del 0,08 %, señalando que el chatbot no fue capaz de responder a ciertas peticiones, pero continuó funcionando sin ningún inconveniente debido a que fue un porcentaje muy pequeño. Con el método POST se inició la simulación ingresando 100 usuarios para 100 peticiones y se observó que no hubo ningún tipo de error, los tiempos promedios para atender las peticiones fueron de 37.798 ms, con un rendimiento de 46 peticiones por minuto. Al aumentar el número de repeticiones a 4 ciclos para llegar atender 400 peticiones, se produjo un error del 78 % indicando que la mayoría de solicitudes no tuvieron respuesta por parte de agente conversacional, e incluso llegando a que el servidor donde estaba aloja la aplicación se reiniciara.

Conclusiones

La búsqueda bibliográfica es una técnica muy efectiva, ya que a través de esta se pudieron obtener 13 artículos científicos donde se mencionaban las tecnologías y herramientas que se podían usar para realizar el ajuste del modelo BERT y por ende ser también usadas para la versión ligera que se escogió DistilBERT. Se pudo conocer, que el lenguaje principal utilizado para llevar a cabo el ajuste es Python, junto con las librerías de PyTorch y Transformers. Además, de que se debe usar como recurso mínimo una GPU con memoria RAM de 16 GB.

Seguir la metodología XP para el desarrollo del agente conversacional fue muy importante, ya que esta permitió dividir el proyecto en diferentes fases, siendo la primera la más esencial, pues es donde se llevó a cabo el entrenamiento de la red neuronal que se utilizaron para el desarrollo del chatbot además de crear las historias de usuario sobre el funcionamiento del mismo, y a partir de ello ejecutar las fases restantes de la metodología.

Las pruebas realizadas con la herramienta ApacheJMeter permitieron conocer las limitaciones del agente conversacional, donde se pudo observar que es capaz de atender hasta 4.800 peticiones de forma simultánea mediante el método GET con un porcentaje muy pequeño de errores (0,08 %) permitiendo que siga funcionando sin problemas. Sin embargo, al usar el método POST y someter el bot a una cantidad constante de 400 peticiones se produjo un error del 78 %, ya que existió un consumo de memoria y CPU muy elevado, provocando que el chatbot y el servidor que contenía la aplicación deje de funcionar por un periodo corto de tiempo.

Agradecimientos

Un agradecimiento a la Universidad Nacional de Loja, a los estudiantes y docentes de la Carrera de Ingeniería en Sistemas/Computación, por su colaboración en el presente trabajo.

Contribuciones de los Autores

Conceptualización: LPR; metodología: LPR; análisis formal: LPR.; investigación: LPR; recursos: LPR; redacción — preparación del borrador original: LPR; redacción — revisión y edición: RFD; visualización: LPR; supervisión: RFD; administración de proyecto: LPR; adquisición de financiamiento para la investigación: LPR. Todos los autores han leído y aceptado la versión publicada del manuscrito.

Leonardo Paredes-Rivas: LPR, Roberth Figueroa-Díaz.

Financiamiento

El presente estudio fue financiado por sus autores.

Referencias

Al-Besher, A., Kumar, K., Sangeetha, M., y Butsa, T. (2022). BERT for Conversational Question Answering Systems Using Semantic Similarity Estimation. *Computers, Materials and Continua*, *70*(3), 4763–4780. doi: 10.32604/cmc.2022.021033

Annamoradnejad, I., Fazli, M., y Habibi, J. (2020). Predicting Subjective Features from Questions on QA Websites Using BERT. *2020 6th International Conference on Web Research, ICWR 2020*, 240–244. doi: 10.1109/ICWR49608.2020.9122318

Bagwe, R., y George, K. (2020). Automatic Numerical Question Answering on Table using BERT-GNN. *2020 11th IEEE Annual Ubiquitous Computing, Electronics and Mobile Communication Conference, UEMCON 2020*, 118–124. doi: 10.1109/UEMCON51285.2020.9298028

Bathija, R., Agarwal, P., Somanna, R., y Pallavi, G. (2020). Guided Interactive Learning through Chatbot using Bidirectional Encoder Representations from Transformers (BERT). *2nd International Conference on Innovative Mechanisms for Industry Applications, ICIMIA 2020 - Conference Proceedings*(Icimia), 82–87. doi: 10.1109/ICIMIA48430.2020.9074905

Chan, Y., y Fan, Y. (2019). A Recurrent BERT-based Model for Question Generation. *MRQA@EMNLP 2019 - Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, 154–162. doi: 10.18653/v1/d19-5821

Devlin, J., Chang, M.-W., Lee, K., y Toutanova, K. (2018). *Bert: Pre-training of deep bidirectional transformers for language understanding.* arXiv. doi: 10.48550/ARXIV.1810.04805

He, Y., Zhu, Z., Zhang, Y., Chen, Q., y Caverlee, J. (2020). Infusing Disease Knowledge Into BERT for Health Question Answering, Medical Inference and Disease Name Recognition. *EMNLP 2020 - 2020 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, 4604–4614. doi: 10.18653/v1/2020.emnlp-main.372

Nikas, C., Fafalios, P., y Tzitzikas, Y. (2020). Two-stage Semantic Answer Type Prediction for Question Answering Using BERT and Class-specificity Rewarding. *CEUR Workshop Proceedings*, *2774*, 19–28.

e-ISSN: 1390-5902
CEDAMAZ, Vol. 12, No. 2, pp. 174–182, Julio–Diciembre 2022
DOI: 10.54753/cedamaz.v12i2.1686

Patel, N., Parikh, D., Patel, D., y Patel, R. (2019). AI and Web-Based Human-Like Interactive University Chatbot (UNIBOT). *Proceedings of the 3rd International Conference on Electronics and Communication and Aerospace Technology, ICECA 2019*, 148–150. doi: 10.1109/ICECA.2019.8822176

Qu, C., Yang, L., Qiu, M., Croft, B., Zhang, Y., y Iyyer, M. (2019). BERT with history answer embedding for conversational question answering. *SIGIR 2019 - Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1133–1136. doi: 10.1145/3331184.3331341

Sung, C., Ma, T., Dhamecha, T., Reddy, V., Saha, S., y Arora, R. (2019). Pre-training BERT on Domain Resources for Short Answer Grading. *EMNLP-IJCNLP 2019 - 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference*, 6071–6075. doi: 10.18653/v1/d19-1628

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). *Attention is all you need.* arXiv. Descargado de https://arxiv.org/abs/1706.03762 doi: 10.48550/ARXIV.1706.03762

Vucetic, D., Tayaranian, M., Ziaeefard, M., Clark, J., Meyer, B., y Gross, W. (2022). Efficient Fine-Tuning of BERT Models on the Edge. Descargado de http://arxiv.org/abs/2205.01541

Zaib, M., Tran, D., Sagar, S., Mahmood, A., Zhang, W., y Sheng, Q. (2021). BERT-CoQAC: BERT-Based Conversational Question Answering in Context. *Communications in Computer and Information Science*, *1362*, 47–57. doi: 10.1007/978-981-16-0010-4_5

Zhang, N., y Xing, Y. (2021). Questions and Answers on Legal Texts Based on BERT-BiGRU. *Journal of Physics: Conference Series*, *1828*(1). doi: 10.1088/1742-6596/1828/1/012035

Zhu, P., y Hauff, C. (2021). *Evaluating BERT-based Rewards for Question Generation with Reinforcement Learning* (Vol. 1) (n.º 1). Association for Computing Machinery. doi: 10.1145/3471158.3472240